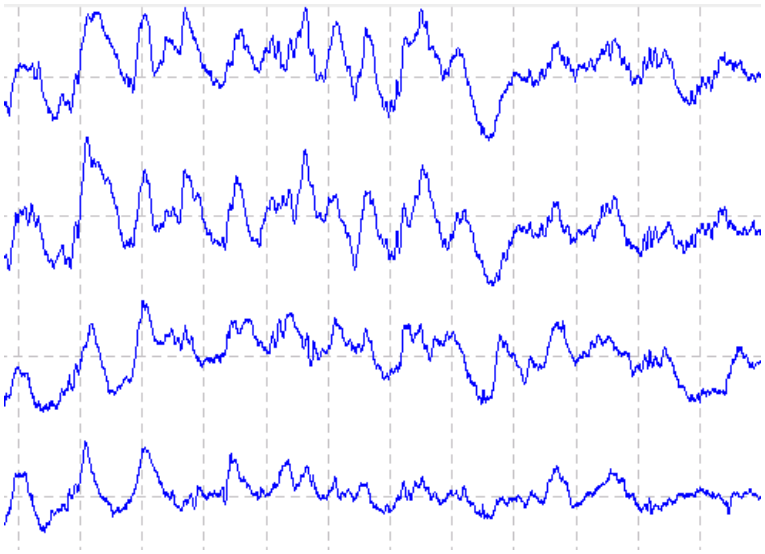
Analyse et modélisation de données médicales pour le diagnostique des troubles du sommeil

Rapport du projet de diplôme



Auteur : Julien Boutillier
Professeur responsable : Dr Laura Elena Raileanu
Mandant : Markus Jaton

Date : 10 Décembre 2007

HEIG-VD
Département du candidat : TIC
Filière : Informatique
Candidat : Boutillier Julien

PROJET DE DIPLOME 2007

Analyse et modélisation de données médicales pour le diagnostique des troubles de sommeil

---

Proposé par : IICT, projet RA&D

Résumé du problème :

A leur actuelle, l'analyse des données EDF (European Data Format) pour les troubles du sommeil, se fait manuellement par l'intermédiaire d'un médecin spécialisé. Cette opération est très conséquente vu la durée de l'enregistrement d'un fichier EDF (~8h).
Le but est de rendre automatique la lecture des données, l'analyse des différents signaux, et l'annotation des résultats trouvés. Ainsi les médecins pourront analyser les résultats d'un simple coup d'œil, et les interpréter selon le cas. Le point délicat est de trouver les algorithmes qui mettront en évidence toutes les anomalies.

Cahier des charges :

Motivation :
La plate-forme IMINET (http://iminet.iict.ch/) constitue une base pour le développement d'applications dans le domaine médical. A ce titre, elle doit fournir des mécanismes permettant l'intégration rapide de nouvelles applications dans le domaine. Si dans un premier temps on s'est essentiellement préoccupé de l'acquisition de données directement à partir du patient, à l'aide de capteurs, de nombreux organes d'acquisition cliniques existent, il semblerait donc logique que les résultats qu'ils délivrent puissent être intégrés à la plate-forme IMINET à des fins d'analyse et de post-traitement (par exemple pour du data mining).

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale

Les divers programmes concernant les données médicales sont principalement des programmes de visualisation, et non pas d'analyse. Ce dernier point est dû à la difficulté d'obtenir une analyse complète d'un fichier, comme c'est normalement le cas avec une analyse manuelle (médecin). Pour approcher cette dernière au maximum, il est nécessaire de réaliser de nombreux testes avec divers modèles statistiques, et de connaitre toutes les anomalies possibles selon les signaux à disposition. L'avantage principal de ce sous-projet est le gain de temps qu'il fera gagner aux médecins, donc à l'application du traitement du patient concerné.

Il n'est bien sûr pas question, dans le cadre d'un projet de diplôme, d'obtenir une analyse complète d'un fichier EDF, cela nécessiterai une importante partie de tests, qui ne serai pas possible de réaliser avec la durée d'un projet de diplôme. L'idée est de prouver qu'il est possible d'extraire les données d'un fichier EDF, de leur appliquer certains algorithmes, et de transmettre les résultats obtenus à la suite de la plate-forme IMINET.

Etapes :
Pour la réalisation de ce projet, voici les différentes étapes mises en évidence :

- Formats de données EDF :
  Il est nécessaire de comprendre le format EDF, pour pouvoir extraire les données, et leur appliquer des divers traitements par la suite.

- Analyse exploratoire de données :
  La connaissance du type de données et des métadonnées fournies est essentielle pour la réalisation du projet. Un exemple de ces données est mis à disposition par Markus Jaton.

- Modélisation de données temporelles :
  Si des corrélations entre différentes séries de données existent, il est important de les identifier, pour détecter un maximum d'anomalies.

- Etude de modèles existants :
  Il est important de connaitre les fonctions, les paramètres, et le résultats retournés par de différents modèles statistiques déjà existants tels que : le simple moving average, le single exponential smoothing, l'incorporating exponential smoothing, l'incorporating seasonal influence.

- Adaptation d'un langage statistique :
  Pour éviter de devoir recréer tous les modèles statistiques utilisés, il serait avantageux de pouvoir appliquer des fonctions d'un langage statistique tel R, S, etc qui ont été déjà implémentées.

- Application des différents modèles :
  Après avoir étudié les différentes modèles, il est nécessaire de les appliquer aux vraies données pour valider leur utilité.

• Validation et test de modèles utilisés.
  Il reste à implémenter les modèles choisis dans l'application finale, et vérifier leur exécution par l'intermédiaire de tests.

• Définition d'un format de résultat :
  Ce format doit être mise en place par le diplômant de ce projet et de celui qui réalise le projet " Internet-based data viewer for XDF, EDF and more ". Ces deux projets sont liés dans la plate-forme IMINET, donc il est impératif qu'ils soient compatibles.

• Développement de l'outil d'analyse :
  Cette étape est la plus conséquente du projet, il s'agit de créer le programme final, avec une interface graphique simple et facile d'utilisation. Ce programme devra être capable de réaliser certaines analyses, et de transmettre le résultat pour la suite du projet IMINET.

Technologies :
La plupart des sous-projets de la plate-forme IMINET qui sont déjà réalisés, sont codés avec la technologie Java, il est important de continuer dans ce sens, ceci pour une complète adaptation des différentes parties du projet IMINET. Il est bien sûr possible de faire appel à d'autre langage depuis Java, comme mentionné ci-dessus avec un langage statistique (R, S, etc).

Le candidat :

Boutillier Julien

La professeure responsable :

Raileanu Laura Elena

Le chef du département TIC :

Eduardo Sanchez

Yverdon-les-Bains, le 19 septembre 2007

Abstract

Pour le moment, l'analyse de données médicales pour le diagnostique des troubles du sommeil est faite à l'aide de quelques analyses existantes, mais principalement par l'intermédiaire des medecins spécialisés. Ce projet offre la possibilité de créer et d'appliquer des analyses sur les données médicale EDF (European Data Format) voulues.

Pour ce faire, il a fallu extraire le données médicales des fichiers EDF, pour qu'elles soyent facilement manipulables. Ainsi il est possible d'accéder à l'interval du signal choisi à l'échantillon prêt, et de lui appliquer tous les calculs désirés.

Pour éviter de devoir réécrire les algorithmes déjà existants, une connexion au serveur Rserve permet d'appeler les multiples algorithmes disponibles sous R. Cette connexion offre de grandes possibilités pour l'analyse des données EDF.

Pour pouvoir utiliser de manière simple ce programme, une interface graphique claire et facile à utiliser a été mise en place pour que les médecins s'y adaptent rapidement. En ce qui concerne les analyses réalisées, elles ont prouvées qu'il est possible d'appliquer des analyses diverses sur les données EDF.

Lors du projet, il a été possible de visiter le centre du sommeil du CHUV et de discuter avec le Dr Heinzer, qui est le médecin responsable de ce centre. Ce dernier nous a proposé de mettre en place une analyse de détection de l'abaissement du pouls. Cette analyse est réalisée en partie, mais avec déjà des résultats concluants.

Les divers résultats obtenus sont des annotations, de durée variable, sur des intervals de signaux analysés. Ce projet faisant partie de la plate-forme IMINET (voir chapitre 9 et [1]), il se doit de transmettre ces résultats pour la suite du la plate-forme, à savoir l'affichage de ces annotations avec les signaux EDF (viewer EDF). Pour cette transmission, un protocole a été mis en place par M. Gavin (projet d'affichage des annotations EDF et mise en réseau) et moi-même. Ainsi toutes les annotations sont gérées par les deux projets, et il n'y a pas de problèmes de compatibilité.

Table des matières

1 Introduction **1**
- 1.1 But et motivation du projet . . . 1
- 1.2 Etapes du projets . . . 2

2 Notions de base de la statistique **3**
- 2.1 Concepts . . . 3
- 2.2 Analyse exploratoire des données . . . 4
  - 2.2.1 Diagrammes . . . 4
  - 2.2.2 Les statistiques élémentaires (numerical summaries) . . . 7

3 Séries Temporelles **10**
- 3.1 Définition . . . 10
- 3.2 Analyse de séries temporelles . . . 10
- 3.3 Effets périodiques . . . 11
- 3.4 Tendance croissante ou décroissante . . . 11
- 3.5 Série stationnaire . . . 12
- 3.6 Modèle ARIMA . . . 12
  - 3.6.1 Intégration . . . 13
  - 3.6.2 Auto-régression . . . 14
  - 3.6.3 Moyenne mobile . . . 14

4 European Data Format (EDF) **16**
- 4.1 Les spécifications . . . 16
- 4.2 Analyse des données EDF . . . 17
  - 4.2.1 Viewer EDF en Java . . . 17
  - 4.2.2 Extraction des données EDF . . . 18
- 4.3 Explication du code réalisé . . . 18
  - 4.3.1 Chargement du fichier EDF . . . 18
  - 4.3.2 Lecture des caractéristiques du fichier . . . 18
  - 4.3.3 Analyse des différents signaux . . . 20

5 Implémentation R sous Java **22**
- 5.1 Recherche . . . 22
- 5.2 La base des fonctions Rserve . . . 23

6 Architecture **24**
- 6.1 Discussion . . . 24
- 6.2 Description . . . 24

7 Interface graphique **26**
7.1 Page d'accueil . . . 26
7.1.1 Chargement du fichier EDF . . . 27
7.1.2 Caractéristiques du fichier EDF . . . 27
7.1.3 Liste et caractéristiques des signaux . . . 27
7.1.4 Choix de l'intervalle . . . 28
7.1.5 Liste pour l'analyse . . . 29
7.1.6 Gestion de la liste pour l'analyse . . . 30
7.1.7 Affichage des signaux . . . 30
7.1.8 Accès à l'analyse . . . 31
7.1.9 Gestion de la fermeture . . . 31
7.1.10 Résultat . . . 32
7.2 Analyse . . . 33
7.2.1 Liste des commandes . . . 34
7.2.2 Ligne de commande . . . 34
7.2.3 Ecran . . . 35
7.2.4 Résultat . . . 35

8 Analyses (Plugins) **38**
8.1 Plugin . . . 38
8.2 Plugins réalisés . . . 39
8.3 Code ARIMA . . . 41
8.4 Ajout d'un plugin . . . 42
8.4.1 Classe *Commande* . . . 42
8.4.2 Codage du plugin . . . 43

9 Transmission du résultat **47**
9.1 Discussion . . . 47
9.2 Fichier XML . . . 47
9.3 Fichier HTML . . . 48
9.4 Arborescence des fichiers . . . 49

10 Mise à jour **52**
10.1 Présentation du code . . . 52
10.2 Améliorations graphiques . . . 52
10.3 Centre du sommeil du CHUV . . . 54
10.3.1 Discussion . . . 54
10.3.2 Analyse de l'abaissement du pouls . . . 54
10.3.3 Avancement . . . 55

11 Perspectives **66**
11.1 Amélioration du fichier HTML . . . 66
11.2 L'étendue d'une analyse . . . 66
11.3 Ajout d'analyses . . . 66
11.4 Gestion d'analyse multi-fichier EDF . . . 67
11.5 Couplage avec un viewer EDF . . . 67

12 Conclusion **69**

Liste des tables **70**

Liste des figures **71**

A Mode d'emploi **74**

B Résultat du projet de M. Gavin **79**

C Journal de travail **83**
C.1 Semaine du 18 au 21 septembre 83
C.2 Semaine du 24 au 28 septembre 2007 83
C.3 Semaine du 1 au 5 octobre 2007 83
C.4 Semaine du 8 au 12 octobre 2007 84
C.5 Semaine du 15 au 19 octobre 2007 84
C.6 Semaine du 22 au 26 octobre 2007 84
C.7 Semaine du 29 octobre au 2 novembre 2007 84
C.8 Semaine du 5 au 9 novembre 2007 84
C.9 Semaine du 12 au 16 novembre 2007 84
C.10 Semaine du 19 au 23 novembre 2007 85
C.11 Semaine du 26 au 30 novembre 2007 85
C.12 Semaine du 3 au 7 décembre 2007 85
C.13 Semaine du 10 au 15 décembre 2007 85

Chapitre 1

Introduction

1.1 But et motivation du projet

Le but du projet est la création d'une application qui permettra :

- de récupérer des données du domaine médical qui sont stockées sous le format EDF.
- d'analyser ces données en utilisant des algorithmes d'analyse de séries temporelles existants.
- d'incorporer des nouvelles méthodes d'analyse de ces données particulières aux spécialistes du domaine (médecin).

Le format EDF (European Data Format) a été créé pour la médecine, plus précisement pour contenir un enregistrement de plusieurs heures, des signaux de différents capteurs placés sur des patients. Un fichier EDF contient toutes les données numérisées de ces signaux, ainsi que les méta-données de l'enregistrement (ID patient, date, durée, etc) et des signaux (nom, fréquence d'échantillonnage, etc).

Ce travail de diplôme devra démontrer la possibilité d'appliquer une série de techniques de modélisation aux données EDF pour venir en aide aux spécialistes du domaine. Il est évident que l'interprétation des résultats obtenus en utilisant ces modèles doit être laissée au spécialiste du domaine (au médecin) et que cette application est un outil d'aide, il ne peut pas substituer le médecin.

Des nombreux travaux de recherche permettant la modélisation de séries temporelles ont été menés par des nombreux chercheurs et les algorithmes crées pour la modélisation ont été implementé en R. C'est un langage pour le calcul statistique et les graphiques. R offre un large éventail d'algorithmes (modélisation linéaire et non-linéaire, des tests statistiques classiques, des analyses de séries temporelles, de classification, de regroupement, ...) et de représentations graphiques. Le langage S est souvent le véhicule de choix pour la recherche en matière de méthodologie statistique, et R fournit un environnement Open Source qui participe à cette activité. C'est pour cela qu'il serait avantageux de pouvoir accéder aux fonctions de R sous Java.

1.2 Etapes du projets

Ce projet de diplôme fournira un programme avec une interface graphique simple et intuitive, permettant de réaliser une analyse sur l'intervalle du signal EDF voulu. Pour la mise sur pied de ce programme plusieurs étapes doivent être réalisées. Tout d'abord, une étude théorique en statistique et sur le format EDF devront être faites, pour mieux comprendre les signaux à analyser et les méthodes existantes. Ensuite l'extraction des données EDF (méta-données et données) et l'adapatation de R sous Java devront être mises en place pour pouvoir réaliser les analyses. Après tout ceci, l'interface graphique pourra être créée, et les tests de divers modèles statistiques seront réalisés. Pour finir, la transmission des résultats sera faite pour garantir la compatibilité du projet avec la plate-forme IMINET (voir chapitre 9 et [1]).

Ces différentes étapes sont réalisées entre le pré-projet de diplôme, du 15 mars au 25 juin 2007 à raison d'une demi-journée par semaine, et le projet de diplôme en lui-même, du 18 septembre au 17 décembre 2007 à raison de cinq jours par semaine. De plus, une défense orale aura lieu avant le 25 janvier 2008.

Toutes ces étapes seront présentées dans le cadre de ce rapport. Les chapitres 2, 3, 4, 5 et 6 présenteront la partie théorique du projet, avec entre autres les séries temporelles, le format EDF et l'adaptation de R sous Java. Ensuite les chapitres 7, 9 et 10 expliqueront la partie pratique du projet, avec entre autres la création de l'interface graphique et la transmission des résultats. On finira par les améliorations possibles et les conclusions, avec respectivement les chapitres 11 et 12.

Chapitre 2

Notions de base de la statistique

Comme ce projet consiste à réaliser des analyses descriptives et à modéliser des séries temporelles (données EDF), il est nécessaire d'entrevoir les principes de base de la statistique. Le contenu de ce chapitre provient du cours de M. Zuber [2].

2.1 Concepts

Pour exploiter au mieux des données, et en tirer des informations pertinentes, on utilise principalement des méthodes statistiques. Cependant la statistique ne se limite pas à l'analyse de données, mais elle s'occupe aussi de leur collecte et de leur interprétation.

La statistique est formée de deux concepts de base : la variabilité et l'incertitude, dont voici leurs définitions :

- *La variabilité représente les variations dues à des changements qui sont souvent imprévisibles. Comme exemple, il y a l'organisme humain qui varie constamment, comme le poids, la taille ou la pression du sang.*
- *L'incertitude représente le doute concernant l'exactitude des données. Comme exemple, il y a l'incertitude d'un appareil de mesure, l'incertitude de l'évolution de données.*

En dépit de la variabilité à laquelle sont soumises les données observées ou mesurées, la statistique souhaite en tirer des conclusions tout en contrôlant totalement le niveau d'incertitude. Cette démarche permet de découvrir des informations quantifiables, pertinentes, mais aussi des tendances.

En statistique, il existe de nombreux termes spécifiques, qui demandent une explication préalable pour bien les comprendre. Les principaux termes utilisés et leur définition sont affichés dans le tableau 2.1.

De plus, la statistique possède deux principales méthodes d'analyse, l'analyse exploratoire des données et l'inférence statistique. Cette dernière est un ensemble de méthodes permettant de tirer des conclusions fiables à partir d'échantillons statistiques (théorie des probabilités), mais elle ne sera pas traitée dans ce rapport.

Termes		Définition
Variable		Type d'information
Variable	quantitative	Quantifie une donnée
	discrète	Ex : nombre d'enfants dans une famille
	continue	Ex : taille d'un être humain
Variable	qualitative	Qualifie une donnée
	catégorique	Ex : le sexe (masculin ou féminin)
	ordinal	Ex : La qualité d'un appareil (bonne, moyenne, mauvaise)
Observation		Valeur d'une variable
Individu		Source des valeurs des variables
Population		Collection d'individus
Données		Collection d'observations de variables
Echantillon		Sous-ensemble d'une population
Paramètre		Caractéristique d'une population
Statistique		Valeur d'une caractéristique calculée à partir d'un échantillon

TAB. 2.1 – Différents termes de la statistique

2.2 Analyse exploratoire des données

L'analyse exploratoire a pour objectif de comprendre, résumer, décrire et représenter les données en vue d'interpréter la réalité, ceci à l'aide des divers graphiques (diagrammes). Le choix du graphique à utiliser dépend, soit du type de variables, du nombre d'observations, soit du message que l'on souhaite transmettre. Ce choix n'est pas facile à faire, d'autant plus si l'on ne connait pas les différents types de graphiques. Voici donc une explication succinte des digrammes les plus utilisés.

2.2.1 Diagrammes

Nous allons décrire les principaux types de diagrammes : en points, branches-et-feuilles, histogramme, boîte à moustache, camembert et nuage de points.

En points (dot plot)

Le diagramme en points permet de détecter des groupes et des disparités, de visualiser la distribution et de relever des observations atypiques. La figure 2.1 montre trois exemples de diagramme en points.

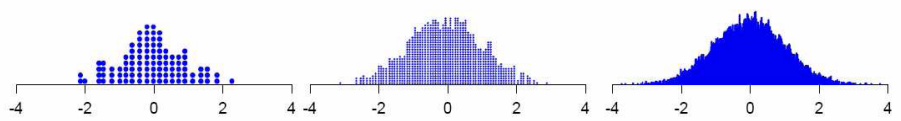


FIG. 2.1 – Diagramme en points

Branche-et-feuilles (stem-and-leaf)

Le diagramme branches-et-feuilles permet de détecter des groupes et des disparités, de visualiser la distribution et de relever des observations atypiques. Cependant si le nombre d'observations dépassent 150, ce diagramme ne permet plus une bonne visualisation. La figure 2.2 montre un exemple de diagramme brances-et-feuilles, avec les observations suivantes : 42, 67, 73, 78, 82, 84, 86, 91, 94, 99.

Test grades	
stem	leaf
4	2
5	
6	7
7	3 8
8	2 4 6
9	1 4 9

FIG. 2.2 – Diagramme branches-et-feuilles

Histogramme (histogram)

L'histogramme permet de détecter des groupes et des disparités, de visualiser la distribution et de relever des observations atypiques. Ce genre de diagramme est utilisé pour des données de type catégoriel, il est très simple à réaliser et facile d'interprétation. La figure 2.3 montre un exemple d'histogramme.

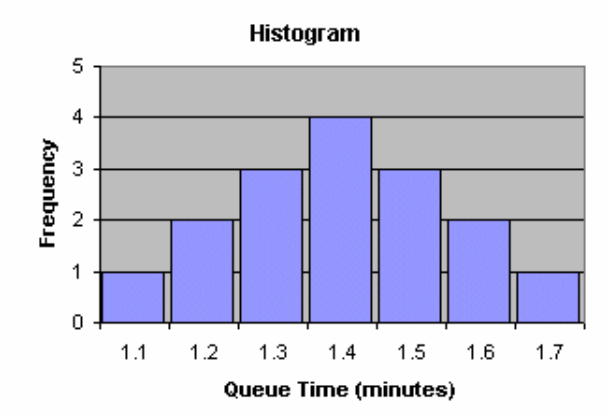


FIG. 2.3 – Histogramme

Boîte à moustache (boxplot)

Ce genre de diagramme permet de représenter des données de type continu, de manière compacte, de comparer la position et la variabilité de différents groupes

d'observations, et de détecter des observations atypiques. La figure 2.4 montre un exemple de boîte à moustache.

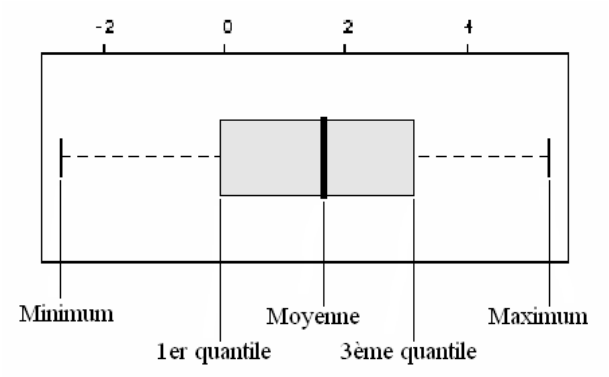


FIG. 2.4 – Boîte à moustache

Camembert (pie chart)

Ce genre de diagramme permet de détecter des groupes et des disparités, et de visualiser la distribution des observations. La figure 2.5 montre un exemple de diagramme camembert.

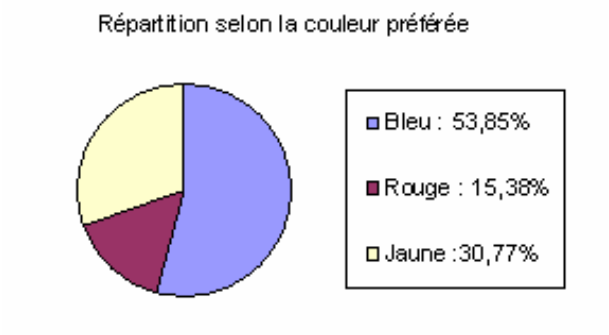


FIG. 2.5 – Diagramme camembert

Nuage de points (scatter plot)

Le graphique le plus simple pour visualiser le lien entre deux variables continues est le « nuage de points ». Pour quantifier ce lien existant entre deux variables, on utilise la covariance définie par la formule 2.1. Les valeurs m_x et m_y représentent respectivement la valeur moyenne des grandeurs X et Y.

$$\frac{1}{n-1}\sum_{i=1}^{n}(x_i - m_x)(y_i - m_y) \tag{2.1}$$

Le degré de dépendance linéaire existant entre les grandeurs X et Y est mesuré par le coefficient de corrélation noté r, et est défini par la formule 2.2.

$$r = \frac{\sum_{i=1}^{n}(x_i - m_x)(y_i - m_y)}{\sqrt{\sum_{i=1}^{n}(x_i - m_x)^2 \sum_{i=1}^{n}(y_i - m_y)^2}} \tag{2.2}$$

Il convient d'accompagner le coefficient de corrélation par un graphique de nuage de points, dont la figure 2.6 en montre un exemple.

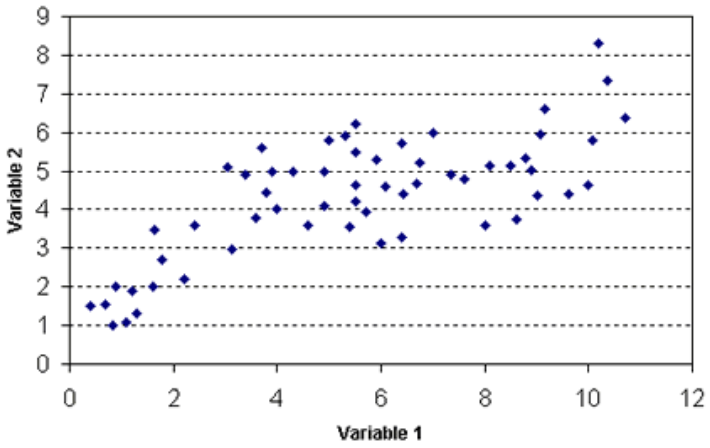


FIG. 2.6 – Graphique en nuage de points

De plus, il existe aussi la matrice de nuage de points (scatter plot matrix). Cette dernière est plus utilisée pour détecter et comprendre les relations existantes entre plusieurs variables. La figure 2.7 montre un exemple d'une matrice en nuage de points.

2.2.2 Les statistiques élémentaires (numerical summaries)

La méthode des statistiques élémentaires consiste à calculer certaines valeurs bien précises, pour mieux comprendre les liens entre les différentes observations. Ces valeurs sont des indicateurs, il en existe de trois types :

- de tendances centrales
- de dispersion
- de forme

Indicateur de tendance centrales

L'indicateur de tendance centrale comme son nom l'indique, informe sur le milieu. En voici quelques uns avec leur déscriptif :

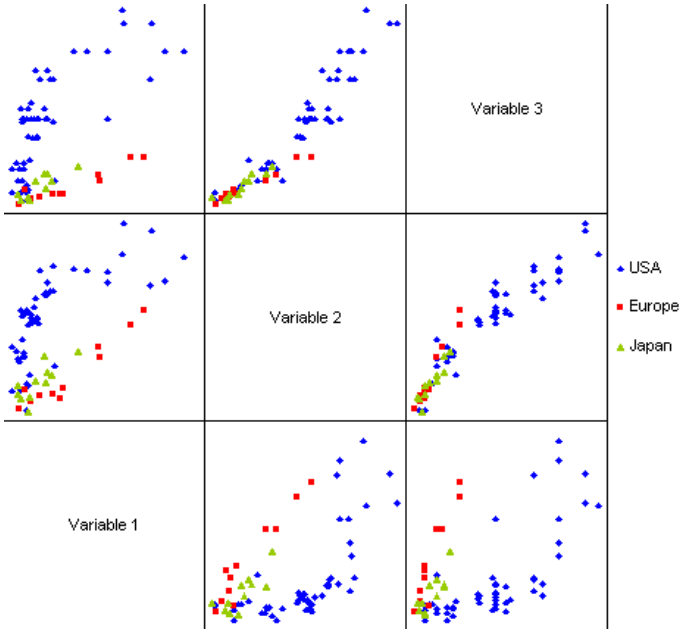


FIG. 2.7 – Matrice de nuage de points

– La **moyenne (mean, average)** est donné par la formule 2.3.

$$m = \frac{x_1 + x_2 + ... + x_n}{n} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{2.3}$$

– La **médiane (median)** coupe l'ensemble des observations en deux parties égales.
– Le **mode (mode)** est la valeur qui est observée le plus grand nombre de fois. Il est possible qu'il y ait deux modes, voire plus.

Indicateur de dispersion

Les indicateurs suivants renseignent sur la variabilité des données :
– L'**étendue (range)**, qui est la différence entre la plus petite et la plus grande observations de l'échantillon.
– L'**écart-type (standard deviation)** est donné par la formule 2.4. La quantité s^2 correspond à la variance de l'échantillon, mais est moins utilisée que l'écart-type, étant donné qu'elle ne s'exprime pas dans la même unité.

$$s = \sqrt{\frac{\sum_{i=1}^{n}(x_i - m_x)^2}{n-1}} \tag{2.4}$$

– L'**étendue interquartile (interquartile range)**. Si l'on veut partager l'échantillon en 4 parties égales, nous devons utiliser 3 quartiles(q(25%), q(50%)=médiane, q(75%)). Il est possible de généraliser ce principe de quartiles en α%-quantiles (q(α%)).

– Le **coefficient de variation (coefficient variation)** est donné par la formule 2.5. Plus ce résultat est petit, plus la série des observations est homogène, concentrée autour de la moyenne.

$$CV = \frac{s}{m} \tag{2.5}$$

Indicateur de forme

Les indicateurs de forme mesurent le degré d'asymétrie des données. En voici les deux principaux :

– L'**asymétrie**, qui est aussi appellée le moment centré d'odre 3, mesure le degré de symétrie présent dans un échantillon. Cette asymétrie est positive s'il y a plus de valeurs à gauche qu'à droite (sur le graphique), et négative si c'est l'inverse. Elle est donnée par la formule 2.6.

$$M_3(x) = \frac{1}{n}\sum_{i=1}^{n}(x_i - m_x)^3 \tag{2.6}$$

– L'**aplatissement**, qui est aussi appellé le moment centré d'orde 4, mesure l'importance de la répartition des valeurs par rapport au centre de symétrie. Si le coefficient d'aplatissement est négatif, cela indique que les queues comptent moins d'observations que dans une distribution normale. Et inversement, s'il est positif, cela indique que les queues comptent plus d'observations que dans une distribution normale. Le coefficient d'aplatissement est donné par la formule 2.7.

$$M_4(x) = \frac{1}{n}\sum_{i=1}^{n}(x_i - m_x)^4 \tag{2.7}$$

Chapitre 3

Séries Temporelles

Maintenant que les notions de base de la statistique sont présentées, nous allons analyser plus en détail les séries de données en fonction du temps, plus communément appellées les séries temporelles. Tout d'abord, on présente les différentes tendances possibles d'une série temporelle (effet saisonnier, tendance croissante ou décroissante), et ensuite une petite explication sur les séries stationnaires. Pour la rédaction de cette partie, le livre de Stephan Morgenthaler a été pris comme référence, voir [3]. Pour finir, on explique les différentes parties du modèle ARIMA, avec leurs équations respectives, en se basant sur la référence [4].

3.1 Définition

Une série temporelle est une série d'observations d'une caractéristique d'un système en fonction du temps, effectuées à un rythme régulier (ex : chaque seconde). On notera les valeurs d'une telle série y_1, y_2, ..., y_t, ..., y_n. *La propriété qui distingue ces données des autres données rencontrées en statistique, est la dépendance stochastique. Cela signifie que la valeur* y_t *dépend fortement de sa valeur précédente* y_{t-1}, *et un peu moins fortement de* y_{t-2}, *etc. Une grande partie de l'analyse des séries temporelles consiste à décrire et à exploiter cette dépendance.*

3.2 Analyse de séries temporelles

La première analyse réalisée en général, est une représentation graphique ou de la statistique simple. L'objectif principal est l'identification et la description des structures contenues dans les données de la série temporelle. Ces structures peuvent être la tendance de la variable y_t en fonction du temps t, un effet saisonnier tel qu'un cycle hebdomadaire ou annuel, les dépendances entre observations voisines, etc. La prévision au-delà de la dernière observation y_n, est souvent le but principal d'une telle analyse statistique. Pour estimer l'observation suivant y_{n+1}, il est naturel de penser à la moyenne de toutes les observations précédentes, ou à la dernière valeur y_n. Ces premières idées sont un peu « légères », surtout si notre série possède une relation avec le temps t, comme une croissance continue, ou un effet périodique. Une série ne contenant aucun de ces effets est dite **stationnaire**.

$(y_1,y_2,y_3,...)$	y_1	y_2	...	y_d	m_1
	y_{d+1}	y_{d+2}	...	y_{2d}	m_2
	y_{2d+1}	y_{2d+2}	...	y_{3d}	m_3
	...	...	...	...	...
effet périodique	$\hat{s}_1$	$\hat{s}_2$	...	$\hat{s}_d$	moyennes des cycles

$\hat{s}_l$ = moyenne de la l^e colonne - moyenne des cycles
m_k = moyenne de la k^e ligne

TAB. 3.1 – Principe pour détecter un effet périodique

3.3 Effets périodiques

La caractéristique la plus prononcée d'une série temporelle est souvent la présence d'une onde périodique, dont la période est connue. Par exemple, une année dans une série mensuelle, ou une semaine dans une série journalière. Un tel effet est dit **effet saisonnier**. Un moyen simple pour éliminer un effet périodique dans une série d'observations est de remplacer la série observée par une série filtrée. Le filtre numérique que l'on doit utiliser dans ce cas est une **moyenne mobile**. Ce principe est simple, il consiste à remplacer la série y_t par la moyenne calculée x_t exactement sur une période, et ainsi l'effet périodique est remplacé par la valeur moyenne. La formule 3.1 montre l'équation pour un effet périodique toutes les 13 valeurs (comme exemple).

$$x_t = \frac{(y_{t-6} + y_{t-5} + ... + y_t + ... + y_{t+5} + y_{t+6})}{12} \tag{3.1}$$

Pour mieux comprendre, il suffit de prendre une série parfaitement périodique est d'appliquer la formule ci-dessus à chaque valeur. Le résultat obtenu est zéro pour toutes les valeurs, ainsi l'effet périodique a disparu.

Une méthode alternative, qui fonctionne même en présence d'une faible tendance périodique, consiste à arranger la série sous forme d'un tableau dont les lignes correspondent aux cycles successifs. Si la période vaut d, alors une série qui commence au début d'un cycle est transformée en tableau et l'effet saisonnier est estimé comme le montre le tableau 3.1.

La suite $\hat{s}_l$ est une estimation de l'effet saisonnier, tandis que m_l représente la tendance de croissance et de décroissance de la série. La série filtrée est obtenue en soustrayant chaque valeur par l'effet saisonnier $\hat{s}_l$ correspondant à la colonne de la valeur.

3.4 Tendance croissante ou décroissante

Une autre caractéristique prononcée d'une série temporelle est la présence d'une tendance croissante ou décroissante de l'espérance des observations. Une telle tendance peut être extraite des données soit par la méthode des **moindres carrés**

t	Série	1ère différence	2ème différence
1	y_1		
		y_2-y_1	
2	y_2		$(y_3$-$y_2)$-$(y_2$-$y_1)$=y_3-2y_2+y_1
		y_3-y_2	
3	y_3		y_4-2y_3+y_2
		y_4-y_3	
4	y_4		y_5-2y_4+y_3
		y_5-y_4	
5	y_5		y_6-2y_5+y_4
		y_3-y_2	
6	y_6		

TAB. 3.2 – Calcul de la 1ère et 2ème différence

(pour plus d'information voir [5]), soit par application d'un filtre différentiel. Cette méthode consiste à calculer la différence entre deux valeurs successives de la série, comme le montre la formule 3.2.

$$x_t = y_t - y_{t-1} \tag{3.2}$$

Ainsi le filtre différentiel peut éliminer une croissance linéaire, et même une croissance quadratique, en calculant la différence de la différence, appelé la deuxième différence, et ainsi de suite, comme le montre le tableau 3.2.

En utilisant les méthodes d'élimination d'effets périodiques et de tendances, une série temporelle non-stationnaire peut être transformée en une série stationnaire.

3.5 Série stationnaire

Pour prédire de manière plus détaillée les valeurs suivantes y_{n+1}, y_{n+2},... d'une série temporelle, il faut utiliser la théorie des processus stochastiques stationnaires. Une suite de variables aléatoires y_1, y_2, y_3,... est dite stationnaire si toutes les sous-séquences d'une longueur l quelconque ont la même distribution. Pour une suite stationnaire, les vecteurs (Y_1, Y_2, Y_3, ..., Y_l) et (Y_k, Y_{k+1}, ..., Y_{k+l}), suivent la même loi pour tout l et k. La stationnarité montre également que toutes les paires (Y_1, Y_{k+1}), ..., (Y_t, Y_{k+t}), ... ont la même loi bivariée, voir [5].

Pour mieux comprendre une série temporelle, il peut s'avérer utile d'avoir à disposition des modèles statistiques pour permettre de comprendre les liens entre les observations de la série. Le modèle ARIMA est parmi les plus connus de ce genre de modèle, voici donc un explicatif de ce modèle.

3.6 Modèle ARIMA

Les modèles comme ARIMA (« Auto-Regressive - Integrated - Moving Average ») cherche à déterminer chaque valeur de la série en fonction des valeurs qui la précèdent (y_t = f(y_{t-1}, y_{t-2}, ...)).

Les processus auto-régressifs supposent que chaque point peut être prédit par la somme pondérée d'un ensemble de points précédents, plus un terme aléatoire d'erreur.

Le processus d'intégration suppose que chaque point présente une différence constante avec le point précédent.

Les processus de moyenne mobile supposent que chaque point est en fonction des erreurs entachant les points précédents, plus sa propre erreur.

Un modèle ARIMA est étiqueté comme modèle ARIMA (p,d,q), dans lequel :

- p est l'ordre des termes auto-régressifs
- d est l'ordre des intégrations
- q est l'ordre des moyennes mobiles.

3.6.1 Intégration

L'estimation des modèles ARIMA suppose que l'on travaille sur une série stationnaire. Ceci signifie que la moyenne de la série est constante dans le temps, ainsi que la variance. La meilleure méthode pour éliminer toute tendance est de différencier, c'est-à-dire de remplacer la série originale par la série des différences adjacentes. Une série temporelle qui a besoin d'être différenciée pour atteindre la stationnarité est considérée comme une version intégrée d'une série stationnaire (d'où le terme Integrated).

La correction d'une non-stationnarité en terme de variance peut être réalisée par des transformations de type logarithmique (si la variance croît avec le temps) ou à l'inverse exponentielle. Ces transformations doivent être réalisées avant la différenciation.

Une différenciation d'ordre 1 suppose que la différence entre deux valeurs successives de y est constante, comme le montre la formule 3.3.

$$y_t - y_{t-1} = \mu + \epsilon_t \tag{3.3}$$

μ est la constante du modèle, et représente la différence moyenne en y. Un tel modèle est un ARIMA (0,1,0). Il peut être représenté comme un accroissement linéaire en fonction du temps. Si μ est égal a 0, la série est stationnaire.

Les modèles d'ordre 2 travaillent non plus sur les différences brutes, mais sur les différences de différence. La seconde différence de y au moment t est égale à $(y_t - y_{t-1}) - (y_{t-1} - y_{t-2})$, c'est-à-dire à $y_t - 2y_{t-1} + y_{t-2}$.

Un modèle ARIMA (0,2,0) obéira à l'équation de prédiction 3.4 ou 3.5

$$y_t - 2y_{t-1} + y_{t-2} = \mu + \epsilon_t \tag{3.4}$$

$$y_t = \mu + 2y_{t-1} - y_{t-2} + \epsilon_t \tag{3.5}$$

3.6.2 Auto-régression

Les modèles auto-régressifs supposent que y_t est une fonction linéaire des valeurs précédentes, comme l'équation 3.6 le montre.

$$y_t = \mu + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \phi_3 y_{t-3} + \epsilon_t \tag{3.6}$$

Littérairement, chaque observation est constituée d'une composante aléatoire (choc aléatoire ϵ) et d'une combinaison linéaire des observations précédentes. ϕ_1, ϕ_2 et ϕ_3 sont les coefficients d'auto-régression.

A noter que cette équation porte soit sur les données brutes, soit sur les données différenciées si une differenciation a été nécessaire. Pour un modèle ARIMA (1,1,0) on aura la formule 3.7 ou 3.8.

$$y_t - y_{t-1} = \mu + \phi(y_{t-1} - y_{t-2}) + \epsilon_t \tag{3.7}$$

$$y_t = \mu + y_{t-1} + \phi(y_{t-1} - y_{t-2}) + \epsilon_t \tag{3.8}$$

Notez qu'un processus auto-régressif n'est stable que si les paramètres sont compris dans un certain interval ; par exemple, s'il n'y a qu'un paramètre auto-régressif, il doit se trouver dans l'interval $-1 < \phi_1 < +1$. Dans les autres cas, les effets passés s'accumulent et les valeurs successives des x_t se déplacent infiniment vers l'avant, ce qui signifie que la série n'est pas stationnaire.

3.6.3 Moyenne mobile

Les modèles à moyenne mobile suggèrent que la série présente des fluctuations autour d'une valeur moyenne. On considère alors que la meilleure estimation est répresentée par la moyenne pondérée d'un certain nombre de valeurs antérieures (ce qui est le principe des procédures de moyennes mobiles utilisées pour le lissage des données). Ceci revient à considérer que l'estimation est égale à la vraie moyenne, auquelle on ajoute une somme pondérée des erreurs ayant entâché les valeurs précédentes, comme le montre l'équation 3.9.

$$y_t = \mu - \theta_1 \epsilon_{t-1} - \theta_2 \epsilon_{t-2} - \theta_3 \epsilon_{t-3} + \epsilon_t \tag{3.9}$$

Littérairement, chaque observation est composée d'une composante d'erreur aléatoire (choc aléatoire ϵ) et d'une combinaison linéaire des erreurs aléatoires passées. θ_1, θ_2 et θ_3 sont les coefficients de moyenne mobile du modèle.

Comme précédemment cette équation porte soit sur les données brutes, soit sur les données différenciées si une différenciation a été nécessaire. Pour un modèle ARIMA (0,1,1) on aura la formule 3.10 ou 3.11 :

$$y_t - y_{t-1} = \mu + \theta \epsilon_{t-1} + \epsilon_t \tag{3.10}$$

$$y_t = \mu + y_{t-1} + \theta \epsilon_{t-1} + \epsilon_t \tag{3.11}$$

Un modèle de moyenne mobile correspond a des séries exhibant des fluctuations aléatoires autour d'une moyenne variant lentement. Plutôt que de prendre comme précédemment la valeur précédente comme prédicteur, on utilise une moyenne de quelques observations précédentes, de manière à éliminer le bruit, et estimer plus précisément la moyenne locale.

Cette logique correspond au **lissage exponentiel simple**, qui considère chaque observation comme la résultante d'une constante b et d'un terme d'erreur ϵ, soit l'équation 3.12.

$$y_t = b + \epsilon_t \tag{3.12}$$

La constante b est relativement stable sur chaque segment de la série, mais peut se modifier lentement au cours du temps. Si ce modèle est approprié, l'une des manières d'isoler la réelle valeur de b, et donc la partie systématique ou prévisible de la série, consiste à calculer une sorte de moyenne mobile, ou les observations courantes et immédiatement précédentes (« les plus récentes ») ont une pondération plus forte que les observations plus anciennes. C'est exactement ce que fait un lissage exponentiel simple, ou les pondérations les plus faibles sont affectées exponentiellement aux observations les plus anciennes. La formule 3.13 est spécifique du lissage exponentiel simple (noter que la valeur 1-α correspond au θ des équations précédentes).

$$y_t = \alpha\hat{y}_t - (1-\alpha)y_{t-1} \tag{3.13}$$

On peut également envisager des modèles mixtes : par exemple un modèle ARIMA (1,1,1) aura l'équation de prédiction 3.14.

$$y_t = \mu + y_{t-1} + (y_{t-1} - y_{t-2}) - \theta_1\epsilon_{t-1} + \epsilon_t \tag{3.14}$$

Après avoir présenté les différentes parties du modèle ARIMA, avec leur équation et leurs explications, nous allons passé au l'analyse du format EDF. Ceci pour comprendre comment extraire les données d'un fichier EDF.

Chapitre 4

European Data Format (EDF)

Le format EDF, Format de Données Européennes (FED) en français, est un format simple et flexible organisé en séries temporelles, pour l'échange et le stockage des signaux biologiques multicanaux. Il a été développé par quelques ingénieurs européens du domaine médical, de 1987 à 1992, date de son édition dans l'électroencéphalographie et la neurophysiologie.

Dans ce chapitre on présente le format EDF avec ses différentes spécifications, le code réalisé pour l'extraction de ces données, et le résultat obtenu. La première partie de ce chapitre se base sur le site du format EDF et EDF+, voir [6].

4.1 Les spécifications

Un fichier EDF contient un enregistrement polygraphique numérisé non interrompu, ceci signifie qu'un fichier EDF contient différents signaux enregistrés durant plusieurs heures.

Ce type de fichier contient une partie d'en-tête, suivi des signaux. Dans l'en-tête se trouvent les informations pour identifier le patient, ainsi que les caractéristiques techniques des signaux. Plus en détail, la liste suivante indique toutes les informations présentes dans l'en-tête d'un fichier EDF :

- Le nom du fichier
- La version du format des données
- L'identifiant du patient
- L'identifiant de l'enregistrement
- Le nombre d'enregistrement
- Le nombre de signaux
- La durée d'un enregistrement
- La durée totale
- Le nombre total d'échantillon par enregistrement
- La date de l'enregistrement

Un fichier EDF est coupé en plusieurs enregistrements ($\sim 3000 = 8\text{h}20$ d'enregistrement) qui durent 10 secondes. Chacun de ces enregistrements contient un nombre d'échantillons équivalent à l'addition du nombre d'échantillons de chaque signal.

Comme exemple, si un fichier EDF contient 30 signaux avec une fréquence

d'échantillonnage identique de 20 Hz, cela signifie que le nombre d'échantillons de chaque signal pendant 10 secondes (un enregistrement) est de $10 * 20 = 200$. Donc le nombre total d'échantillons par enregistrement (par 10 secondes) est $30 * 200 = 6000$. Mais en réalité les signaux n'ont pas la même fréquence d'échantillonnage, donc pas tous le même nombre d'échantillons par enregistrement, mais le principe est le même.

Après l'en-tête du fichier, on trouve les signaux les uns après les autres, avec pour chacun d'eux un en-tête avec leurs caractéristiques, dont voici la liste complète :

- Le nom du signal
- Le nombre d'échantillon par enregistrement
- Le pré-filtre appliqué (champ vide, si aucun filtre n'est appliqué)
- Le type de capteur utilisé
- La fréquence d'échantillonnage
- La valeur numérique maximum
- La valeur numérique minimum
- L'unité physique des données
- La valeur physique maximum
- La valeur physique minimum

Toutes ces informations permettent de mieux connaître les données, et de mieux les traiter. Par exemple, il faut faire attention au fait que les données récupérées ont des valeurs numériques, une conversion en valeurs physiques doit être réalisée grâce aux informations du signal correspondant.

Il existe une évolution du format EDF, appellée EDF+ (sorti en 2002). Au lieu d'appeler cela une évolution, il serait plus juste de parler de protocole EDF+. La grande différence est la mise en place de conventions concernant toutes les caractéristiques du fichier et des signaux. De plus des nouveaux champs ont été ajoutés pour mieux préciser les circonstances de l'enregistrement, et le patient « mesuré ». A part ce plus que fourni EDF+, l'enregistrement et le résultat sont similaires à celui d'EDF, et le résultat aussi. Pour plus d'informations sur l'analyse du format EDF+, veuillez consulter [6].

4.2 Analyse des données EDF

Le but de ce sous-chapitre est de présenter comment obtenir les données EDF sous forme d'un tableau de données, et leurs caractéristiques, sous forme de texte. Ceci est réalisé pour pouvoir effectuer par la suite des analyses sur ces données.

4.2.1 Viewer EDF en Java

Lors des recherches concernant l'extraction des données d'un fichier EDF, les codes sources d'un viewer EDF en Java (jEDF) ont été trouvés. Le sources du jEDF ont été réalisés par Nizar Kerkeni [7]. Ce programme permet d'afficher les signaux

contenus dans un fichier EDF, de réaliser une FFT[1] sur l'interval d'un signal choisi, d'accéder facilement à la partie de l'enregistrement voulue, et d'analyser chaque signal indépendamment. Ces codes sources vont faciliter l'extraction des données EDF.

4.2.2 Extraction des données EDF

En analysant les codes sources du viewer EDF, il a été possible de reprendre une partie du code pour extraire les données EDF. Cette partie est dans le répertoire EDF du projet [8]. En utilisant les diverses fonctions reprises du package EDF, il est possible d'obtenir les données de chaque signal dans une matrice ayant le nombre de lignes égal au nombre de signaux, et le nombre de colonnes égal au nombre de données de chaque signal. Il faut bien faire attention au fait que le nombre de données varie selon le signal lu, il est nécessaire de bien initialiser la matrice afin de ne pas perdre de données, ou qu'il y ait des problèmes lors de l'exécution du programme. De plus, les fichiers EDF étant des fichiers de grande taille (100-200Mo), il est nécessaire de lire les parties des signaux qui nous intéressent pour ne pas surcharger inutilement l'espace mémoire alloué.

4.3 Explication du code réalisé

Le code realisé se décompose en différentes parties, dont voici la liste :

- Chargement du fichier EDF
- Lecture des caractéristiques du fichier
- Lecture des caractéristiques des signaux
- Lecture des différents signaux

Il est possible d'effectuer des analyses et des calculs sur les données sélectionnées.

4.3.1 Chargement du fichier EDF

Pour le chargement, l'idée a été reprise du viewer EDF, grâce auquel on arrive à obtenir une variable *EDFFile* qui contient toutes les informations concernant le fichier et ses signaux. Il est donc possible de lire n'importe quelque signal depuis la seconde voulue. Le code se référant à cette partie est affiché à la figure 4.1.

4.3.2 Lecture des caractéristiques du fichier

Comme vu précédemment dans ce chapitre, les diverses informations contenues dans l'en-tête du fichier sont importantes, il est donc intéressant de pouvoir les lire. Pour extraire ces informations, les fonctions de la classe *EDFFile* sont à disposition. Ainsi l'accès à ces informations n'est pas très compliqué à réaliser (figure 4.2).

[1] Fast Fourrier Transform : la transformée de Fourrier rapide utilisée en traitement numérique du signal pour transformer des données temporelles en données fréquentielles.

```java
// Chargement du fichier EDF
public void chargement(File file){
    try {
        edfFile = new EDFFile(file);
        // Obtention du nombre de signaux
        nbSignal = edfFile.getNbSignals();
        signals = new EDFSignal[nbSignal];
        signalData = new EDFSignalData[nbSignal];
        // Chargement des signaux dans les variables
        for(int i=0;i<nbSignal;i++){
            signals[i] = edfFile.getSignal(i);
            signalData[i] = signals[i].getSignalData();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

FIG. 4.1 – Chargement du fichier EDF

```java
// Affiche les caractéristiques du fichier EDF
public void caractFile(){

    System.out.println("Caractéristiques du fichier " +
                       edfFile.getFileName() + " : ");
    System.out.println("\t - Version : " + edfFile.getVersion());
    System.out.println("\t - PatientID : " + edfFile.getPatientID());
    System.out.println("\t - RecordID : " + edfFile.getRecordID());
    System.out.println("\t - NbRecords : " + edfFile.getNbRecords());
    System.out.println("\t - NbSignal : " + edfFile.getNbSignals());
    System.out.println("\t - RecordDuration : " + edfFile.getRecordDuration());
    System.out.println("\t - TotalDuration : " + edfFile.getTotalDuration());
    System.out.println("\t - TotalNumberOfSamplesPerRecord : "
                      + edfFile.getTotalNumberOfSamplesPerRecord() + "\n");
    System.out.println("\tStartTime : " + edfFile.getStartTime() + "\n");
}
```

FIG. 4.2 – Accès aux caractéristiques du fichier EDF

Pour le moment ces informations sont écrites sur la console, pour prouver qu'il est possible de les obtenir. Mais par la suite, il sera possible de les afficher où cela paraît utile. La figure 4.3 montre un exemple de ces caractéristiques.

```
Caracteristiques du fichier Traces.edf :
        - Version : 0
        - PatientID : Markua JATON
        - RecordID : Somnologica data file with signal type and GUID
        - NbRecords : 2384
        - NbSignal : 37
        - RecordDuration : 10
        - TotalDuration : 23840
        - TotalNumberOfSamplesPerRecord : 27030
```

FIG. 4.3 – Exemple des caractéristiques d'un fichier EDF

4.3.3 Analyse des différents signaux

Dans cette partie la lecture des signaux et de leurs caractéristiques sont traités simultanément. Une fonction du package EDF permet de lire un signal en donnant la seconde de départ et le nombre de secondes désiré, et d'obtenir cet interval sous forme d'un tableau. Cette fonction est pratique, mais il serait intéressant de pouvoir obtenir cet interval à l'échantillon prêt, et non pas à la seconde. Ainsi certaines modifications ont été apportées pour permettre de préciser la lecture.

En ce qui concerne l'affichage des données, il suffit de parcourir le tableau résultant, et pour les caractéristiques du signal, c'est aussi simple que celles du fichier. La figure 4.4 montre le code correspondant à ces caractéristiques et à l'affichage des données. Cette fonction n'est utile que pour le pré-projet de diplôme, car lors du projet de diplôme l'affichage des données se fera à l'aide d'un graphique. La figure 4.5 montre le code correspondant à la lecture d'un interval, et la 4.6 est un exemple des caractéristiques d'un signal.

```
// Affiche les caractéristiques du signal
public void caractSignal(int numSignal){

    signal = signals[numSignal];

    System.out.println("Signal " + signal.getLabel() + " : ");
    System.out.println("\tCaractéristiques : ");
    System.out.println("\t - NbSamples : " + signal.getNbSamples());
    System.out.println("\t - Prefiltering : " + signal.getPrefiltering());
    System.out.println("\t - TransducerType : " + signal.getTransducerType());
    System.out.println("\t - SamplingRate : " + signal.getSamplingRate());
    System.out.println("\t - DigMax : " + signal.getDigMax());
    System.out.println("\t - DigMin : " + signal.getDigMin());
    System.out.println("\t - PhysDim : " + signal.getPhysDim());
    System.out.println("\t - PhysMax : " + signal.getPhysMax());
    System.out.println("\t - PhysMin : " + signal.getPhysMin() + "\n");
}

// Affiche les valeurs lues
public void affichage(float[][] d, int numSignal, int startData, int nbData){

    signal = signals[numSignal];

    System.out.println("\tValeurs du signal de la " + startData
                       + "ème à la " + (startData+nbData-1)
                       + "ème donnée : \n");
    System.out.println("\t  N°\t| N° éch.\t : \tValeur\t\tUnité |");
    System.out.println("\t-------------------------------------------------------");
    for (int i=0; i<d[numSignal].length; i++){
        System.out.println("\t  " + i + "\t| Donnée "
                           + (startData+i) + "\t : \t"
                           + d[numSignal][i]
                           + "\t " + signal.getPhysDim() + " |");
    }
    System.out.println();
}
```

FIG. 4.4 – Accès aux caractéristiques du signal et de l'affichage des données du signal

On remarque que les caractéristiques extraites du fichier et des signaux sont bien celles indiquées au début de ce chapitre.

Après avoir vérifié le bon fonctionnement de ces exemples, il est possible d'obtenir n'importe quel interval de n'importe quel signal avec les informations qui y sont liées.

```
// Lecture des données demandées du signal voulu
public float[][] lecture(float [][] d, int numSignal,
                         int startData, int nbData){

    int startSec, nbSec, supplement, cpt;
    float convertion;
    int fEch;
    short[] dataTemp;
    signal = signals[numSignal];

    fEch = (int)signal.getSamplingRate();
    // Caclcul de la plage de secondes à lire
    startSec = startData/fEch;
    supplement = startData % fEch;
    nbSec = ((startData+nbData)/fEch)+1;
    // Caclul le rapport de convertion (Digital => Physique)
    convertion = (float)(signal.getDigMax()/signal.getPhysMax());

    // Lecture des secondes nécessaire
    dataTemp = signalData[numSignal].readSeconds(startSec, nbSec);

    // Convertion et copiage des valeurs demandées dans "data"
    cpt = 0;
    for (int i=supplement; i<(supplement+nbData); i++){
        d[numSignal][cpt] = dataTemp[i]/convertion;
        cpt++;
    }

    return d;

}
```

FIG. 4.5 – Lecture du signal

```
Signal ECG :
        Caracteristiques :
         - NBSamples : 2000
         - Prefiltering :
         - TransducerType : EKG
         - SamplingRate : 200.0
         - DigMax : 32767
         - DigMin : -32767
         - PhysDim : uV
         - PhysMax : 3997.69
         - PhysMin : -3997.69
```

FIG. 4.6 – Exemple de caractéristiques d'un signal

Chapitre 5

Implémentation R sous Java

Dans ce chapitre, nous présenterons comment atteindre les fonctions du logiciel statistique R pour les utiliser sous Java.

5.1 Recherche

Une première approche trouvée consiste à installer JRI (Interface Java/R, qui permet d'exécuter R dans une application Java à l'aide d'un seul thread), en configurant la machine, puis en créant un fichier run.bat. Grâce à ce fichier, il est possible d'exécuter des fichiers Java comprenant des appels de fonctions R. Toutefois, avant de pouvoir utiliser ce fichier run.bat il faut ajouter la librairie *org.rosuda*, qui comprend les fonctions nécessaires à l'appel des fonctions R.

Pour pouvoir exécuter les fichiers pour l'installation de JRI, il faut un programme reconnaissant la commande *sh*, *ls*. Ces commandes sont celles utilisées sous la console de linux. Le programme utilisé s'appelle MSYS, qui n'est qu'une partie de MinGW[1]. Lors de la configuration de JRI, l'exécution du fichier « configure.win » ne s'est pas déroulé correctement. Après avoir réinstaller les différents composants dans l'ordre correct, le problème ne s'est plus posé, et la création du fichier run.bat a pu être lancée. Mais l'exécution du « make » n'a pas abouti, pour des problèmes de configuration de MSYS. A force de tester différentes possibilités, le fichier run.bat a été généré, mais pour des raisons non-résolues, c'est la seul fois que ce fichier a pu être obtenu. Différents tests supplémentaires ont été réalisés, mais aucun résultat n'en ait sorti.

Une autre méthode m'a été conseillée par M. Poulin, qui consiste à utiliser Rserve[2]. Ce dernier permet de se connecter à un serveur, de lui envoyer des fonctions R et d'obtenir le résultat. Pour la simplicité d'utilisation de cette méthode, la première est laissée de côté pour le moment. Pour que ce principe fonctionne, il faut ajouter la librairie *JRClient* qui contient les fonctions nécessaires au fonctionnement de Rserve. Une fois ceci fait, il suffit de taper la ligne suivante dans le code Java, et la connexion est faite.

```
Rconnection c = new Rconnection();
```

[1]Invite de commnade open source, avec la possibilité d'ajouter des packages comme MSYS, permettant la compilation de différents types de langages.

[2]Serveur TCP/IP permettant à d'autres programmes d'utiliser des fonctions de R.

Il faut néanmoins lancer le serveur Rserve avant l'exécution de cette ligne, ceci en exécutant le fichier Rserve.exe. Pour que son exécution se passe correctement, il faut avoir copié les quatre fichiers .dll qui se trouvent dans le répertoire bin du programme R (R.dll, Rblas.dll, Rlapack.dll et Rproxy.dll). Pour exécuter Rserve.exe, il est possible de le faire manuellement à l'aide de la console, ou automatiquement en codant la ligne suivante, en spécifiant le chemin d'accès relatif, ou absolue.

```
Process p = Runtime.getRuntime().exec( "./Outils/Rserve/Rserve.exe" );
```

Dans ce cas, le chemin d'accès est relatif, et le fichier Rserve.exe et placé dans un répertoire Rserve du répertoire Outils présent à la racine du projet.

Après cela, il est possible d'appeler des fonctions R dans un fichier Java, dont voici des exmples dans la partie suivante.

5.2 La base des fonctions Rserve

Les fonctionnalités de base de Rserve se constituent de deux fonctions principales, *Rconnection.assign()* et *Rconnection.eval()*. Pour mieux comprendre voici tout de suite un exemple avec la figure 5.1.

```
double[] data  = ...;

Rconnection c = new Rconnection();

c.assign("data",data);
result = c.eval("mean(data)").asDouble();
```

FIG. 5.1 – Exemple basique des fonctions de Rserve

Dans cet exemple, on voit tout d'abord l'initialisation d'un tableau de valeur, puis la connexion au serveur, comme mentionné ci-dessus. Ensuite, on assigne le tableau à une variable R, ainsi cette dernière peut être utilisée dans une fonction R comme paramètre. Ceci est montré à la dernière ligne de l'exemple, la fonction *Rconnection.eval()*, permet d'évaluer une « requête » R et de retourner une variable de type *REXP*. Cette classe permet de caster la valeur de retour dans différents types Java (*double*, *int*, *double[]*, etc), comme dans l'exemple avec *REXP.asDouble()*.

Chapitre 6

Architecture

L'architecture d'une application joue un rôle très important pour sa flexibilité et sa modularité, mais aussi pour une meilleure compréhension des programmeurs qui devront reprendre ce projet par la suite. Ce chapitre présentera donc les points importants de l'architecture de l'application.

6.1 Discussion

Après discussion avec Mme Raileanu et M. Poulin, il en est sorti qu'il serait intéressant d'utiliser une architecture en plugin. Plus en détail, il s'agit de créer une interface pour exécuter différents plugins de la même manière. Ceci signifie que tous les plugins devront implémenter les mêmes méthodes (retour du résultat, nombre de paramètres, etc). Ainsi la modularité du projet est optimum, et il est facile de l'évoluer.

6.2 Description

Comme mentionné ci-dessus, l'architecture choisie correspond à la gestion de plugins. La figure 6.1 illustre le principe de base.

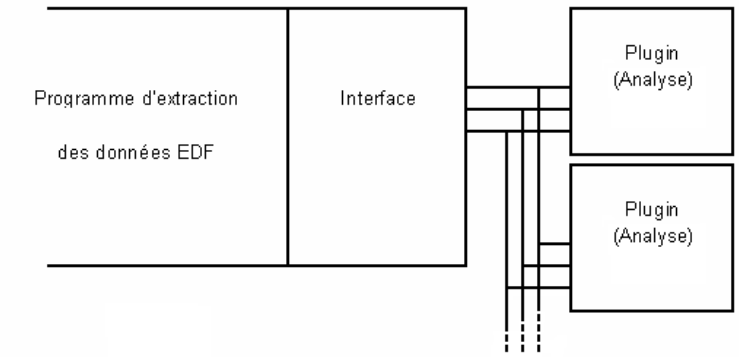


FIG. 6.1 – Illustration de l'architecture du programme

Selon ce schéma tous les plugins doivent implémenter les mêmes fonctions, qui sont déclarées dans l'interface, mais en exécutant chacune une analyse différente. Il faut néanmoins récupérer le résultat du plugin, et comme la méthode est la même pour tous les plugins, il est nécessaire de retourner un type général, ou un type créé particulièrement pour ce genre de résultat. Pour le moment, le choix n'est pas facile à faire, mais lors de la réalisation les besoins seront bien connus et le choix sera plus « sûr ».

Le reste du programme n'a rien de particulier, il s'agit d'une architecture normale de code orienté objet. Le plus gros du projet réside dans l'interface graphique, et ses différentes fonctionnalités, qui sont détaillées dans le chapitre suivant.

Chapitre 7

Interface graphique

Ce chapitre présente les différentes parties que l'interface graphique devra posséder. Plus en détail, il y aura une page principale pour le choix du fichier EDF et des intervals à analyser, ainsi qu'une page d'analyse pour appliquer les algorithmes voulus sur les intervalles sélectionnés. Mais voici tout de suite le détail de ces parties.

7.1 Page d'accueil

Cette partie présente le fichier chargé et ses différents signaux, et permet aussi de choisir l'intervalle des signaux à analyser. Elle doit par conséquent proposer plusieurs fonctions et informations utiles à l'utilisateur, dont voici une liste :

- La possibilité de charger un fichier EDF.
- Les caractéristiques du fichier (nom du fichier, ID du patient, etc).
- La liste de tous les signaux.
- Les caractéristiques des différents signaux (nom du signal, unité, etc).
- La possibilité de choisir l'intervalle de manière précise (à l'échantillon prêt).
- La liste des intervalles des signaux choisis pour l'analyse
- La possibilité de gérer cette liste (ajout, suppression, réinitialisation).
- La possibilité d'afficher l'intervalle des signaux choisis.
- La possibilité d'accéder à la partie analyse.
- Une bonne gestion de la fermeture du programme et de la sauvegarde des résultats.

Cette liste n'est pas forcément exhaustive, mais un refaçonnage complète sera réalisé une fois le prototype terminé. Il permettra d'améliorer la présentation, mais aussi les fonctionnalités et le code lui-même. Cette étape est conséquente, il a fallu par conséquent prévoir une semaine en fin de projet pour la réaliser.

Les explications de chaque élément de la liste précédente ne vont pas rentrer trop dans les détails, surtout du point de vue du code. Pour plus d'information à ce niveau, les codes sources sont à disposition.

7.1.1 Chargement du fichier EDF

Cette fonction nécessite l'ajout d'un menu « Ouvrir » qui charge le fichier EDF sélectionné par l'utilisateur, ceci comme expliqué dans le chapitre 4. Il faut néanmoins faire attention de limiter le choix de l'utilisateur uniquement aux fichiers EDF, pour ceci il est nécessaire de créer un filtre (*FilteExtension* qui hérite de *FileFilter*) pour n'afficher que les fichiers voulus. La figure 7.1 qui met en avance les explications données.

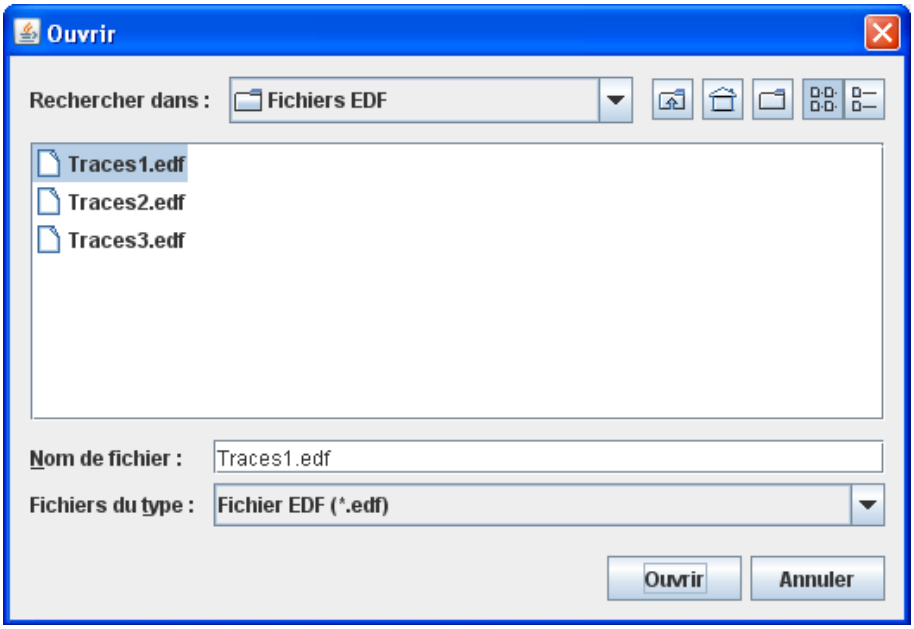


FIG. 7.1 – Choix du fichier EDF

7.1.2 Caractéristiques du fichier EDF

L'accès à ces informations est déjà possible, comme expliqué dans le chapitre 4, il suffit dès lors de les afficher lors de l'ouverture du fichier. Mais quel est le meilleur moyen pour cela ? La première approche trouvée permet d'afficher clairement les différentes informations, grâce à un tableau à deux colonnes, comme le montre l'exemple de la figure 7.2. Cette solution peut toujours être changée, en créant par exemple une fenêtre à part pour l'affichage des informations, mais pour le moment cela convient très bien.

7.1.3 Liste et caractéristiques des signaux

La taille de cette liste peut varier selon les fichiers EDF, il est donc nécessaire de vérifier la taille de la liste des signaux internes au fichier. Ainsi il est possible d'accèder aux diverses informations des signaux, dont leur nom. Grâce à cela la liste des signaux que nous voulons créer est facile à initilialiser. En ce qui concerne les caractéristiques des signaux, il n'est pas concevable de les afficher en totalité à l'écran, vu que le fichier contient en moyenne une trentaine de signaux. Pour résoudre ce problème, il est envisageable d'afficher les caractéristiques d'un seul

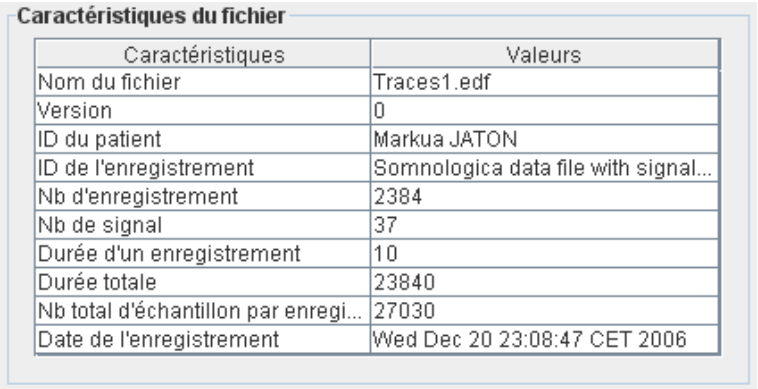


FIG. 7.2 – Affichage des caractéristique du fichier EDF

signal à la fois, dans un tableau comme celles du fichier EDF, et de pouvoir changer de signal facilement. Pour cela, il faut connecter la liste des signaux au tableau des caractéristiques, ainsi un simple clic dans la liste change le tableau instantanément. La figure 7.3 montre un exemple de cette liste et des caractéristiques d'un signal.

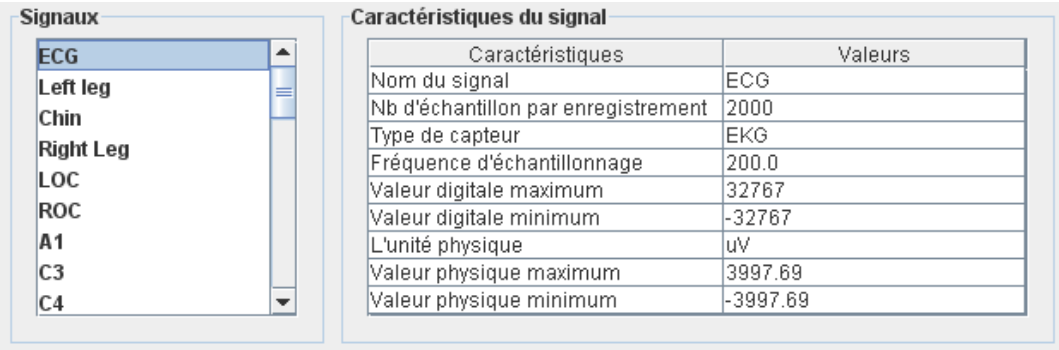


FIG. 7.3 – Affichage de la liste des signaux et des caractéristiques d'un signal

7.1.4 Choix de l'intervalle

Pour cette partie, il est important de trouver un moyen simple et pratique de choisir un intervalle d'un signal, mais la difficulté est dans l'unité choisi. En effet, cette dernière peut être soit des secondes, soit des échantillons. Dans le cas des secondes, l'avantage est de pouvoir facilement accéder à l'intervalle voulu, ceci en ayant regardé précédemment les signaux sur un viewer EDF. De cette manière, le temps de départ et la durée de l'intervalle sont connus. Mais le choix des secondes possède aussi un désavantage, il n'est précis qu'à la seconde, hors le nombre d'échantillon dans une seconde peut varier selon les signaux de 1 éch/sec à 200 éch/sec.

Ainsi l'avantage de la méthode des échantillons est de permettre une précision à l'échantillon prêt, ce qui est très important pour les analyses. En contre partie, il est plus difficile d'obtenir exactement l'intervalle voulu, ceci est principalement dû au fait que chaque signal compte entre 23'840 et 4'768'000 échantillons.

Comme les deux unités présentent un avantage, l'interface laisse l'utilisateur choisir. De plus, il est possible d'afficher l'intervalle choisi pour vérifier si ce dernier correspond bien aux attentes de l'utilisateur. Cette fonction sera plus détaillée dans la partie 7.1.7. La figure 7.4 montre comment on peut choisir l'unité de l'intervalle.

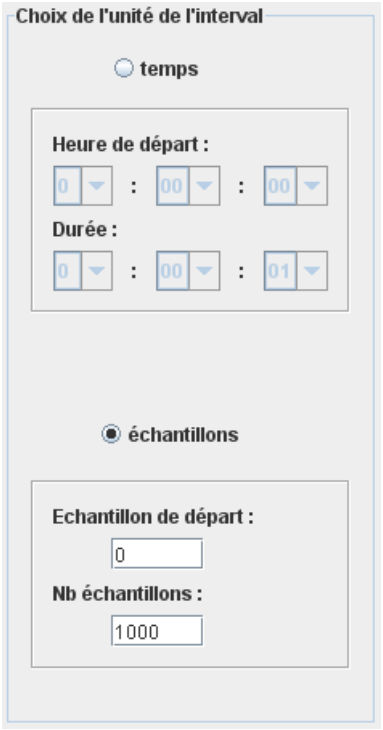


Fig. 7.4 – Choix de l'intervalle

7.1.5 Liste pour l'analyse

Au départ, l'analyse a été réalisée sur un seul intervalle, mais après plusieurs résultats concluants, l'adaptation à plusieurs signaux a été ajoutée. Cette dernière permet de créer une liste de signaux qui sera transmise à la partie analyse. Elle pourra soit appliquer les mêmes analyses à tous les signaux de la liste, soit analyser les divers liens qui pourraient y avoir entre ces signaux (corrélation, similitude spectrale, etc).

Chaque élément de cette liste correspond à un intervalle d'un signal donné, avec le nom du signal, l'index du signal dans le fichier EDF, l'unité des valeurs, la fréquence d'échantillonnage, l'échantillon de départ et le nombre d'échantillons. De plus, pour pouvoir identifier à 100% le signal analysé, il est nécessaire d'avoir le nom du fichier EDF correspondant et l'ID du patient. Pour avoir un élément de ce type, il faut créer un nouvel objet contenant toutes ces informations. Cet objet est décrit à la partie 7.1.8.

Il serait judicieux d'afficher ses informations lorsque l'utilisateur crée sa liste de signaux à analyser, pour qu'il puisse vérifier les derniers signaux ajoutés. Le problème est le même que pour la partie 7.1.3, il n'est pas possible d'afficher les informations

de tous les signaux choisis. Ainsi le meilleur moyen est de prévoir la place pour les informations d'un signal et d'afficher celles qui correspondent au signal choisi dans la liste.

La taille de la liste pour l'analyse est limitée par une constante (pour le moment = 10, car cela est largement suffit pour ce projet) qui peut être changée facilement, en accédant au code.

7.1.6 Gestion de la liste pour l'analyse

Pour gérer une liste comme celle décrite précédemment, il est nécessaire de disposer d'une fonction d'ajout et de suppression d'un élément, et une de réinitialisation de la liste. Cette dernière est utile pour ne pas devoir supprimer tous les éléments un à un, si l'utilisateur désire recommencer à zéro.

La suppression demande une bonne gestion de l'indexation de la liste, pour ne pas poser de problèmes pour la suite de l'exécution. La figure 7.5 montre un exemple de la liste des signaux pour l'analyse, avec ses différentes fonctions.

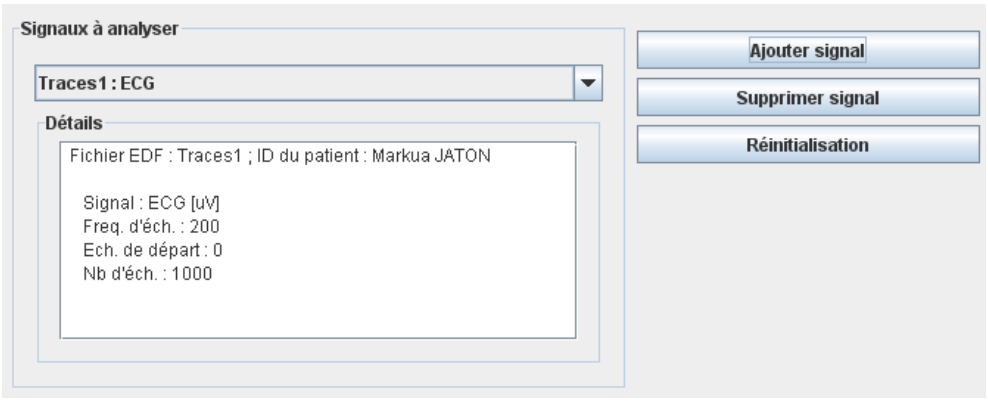


FIG. 7.5 – Exemple de la liste des signaux pour l'analyse

7.1.7 Affichage des signaux

Comme mentionné dans la partie 7.1.4, la possibilté de pouvoir afficher l'intervalle choisi est une fonction indispensable pour ce programme. Pour pouvoir afficher ces parties de série temporelle, il faut une librairie qui permet de créer facilement un graphique, en lui passant les données en paramètres. Le package utilisé s'appelle jfreeChart, il permet de créer toutes types de graphiques, comme par exemple des nuages de points, des histogrammes, etc. Tout ceci avec un ou plusieurs signaux. De plus, ce package offre la possibilité de zoomer sur le graphique et de revenir à l'état initial en un click de souris. La figure 7.6 montre un exemple d'un graphique réalisé à l'aide de jfreeChart.

La fonction d'affichage d'un intervalle se présente sous la forme d'un simple bouton, qui affiche l'intervalle choisi dans la liste des signaux pour l'analyse. Ceci oblige l'utilisateur à insérer l'intervalle avant de l'afficher, mais comme la fonction

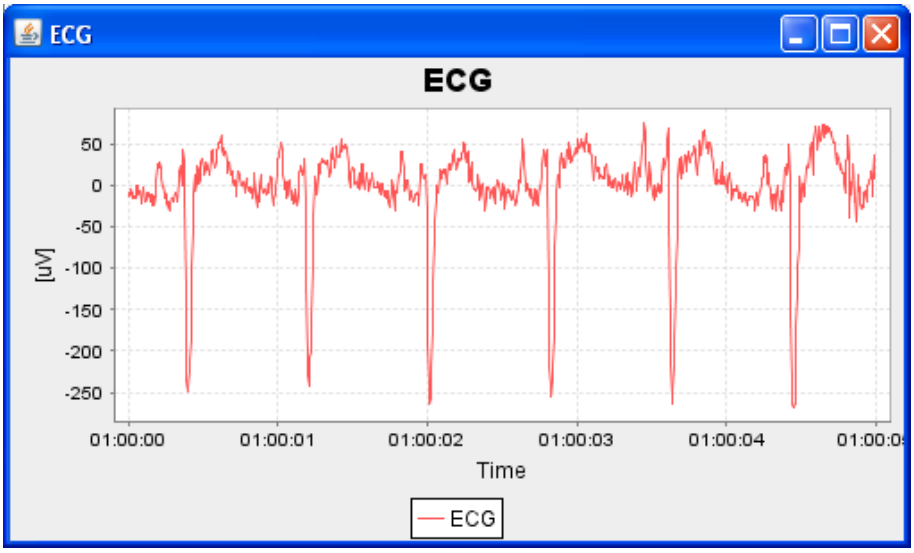


FIG. 7.6 – Exemple de graphique avec jfreeChart

de suppression existe, ceci ne pose pas de problème particulier. De plus, il est tout à fait possible d'afficher plusieurs signaux à la fois, ceci pour comparer deux intervalles différents avant de leur appliquer une analyse commune.

7.1.8 Accès à l'analyse

Pour accéder à la partie analyse, il suffit d'ajouter un simple button, mais qui ne doit pas pouvoir être activé lorsque la liste de signaux à analyser est vide. De plus, il faut à tout moment pouvoir transmettre cette liste, ainsi que les diverses paramètres utiles pour l'analyse. Il est bien sûr nécessaire d'avoir les valeurs du signal dans cet intervalle, mais pas seulement, il faut aussi avoir assez d'informations sur ce signal pour pouvoir l'identifier à coup sûr lors de la lecture des résultats. Pour ceci, il est intéressant de créer une classe « Data » qui représente un intervalle d'un signal, avec les informations données dans la partie 7.5.

Ainsi pour transmettre toute la liste des signaux à analyser et leurs informations, il suffit de créer un tableau de la classe « Data », en insérant un élément à chaque intervalle ajouté. Pour mieux comprendre cette classe, la figure 7.7 affiche le code correspond.

7.1.9 Gestion de la fermeture

Pour ce genre de programme le résultat final est la sauvegarde des différentes analyses effectuées et des graphiques obtenus. Comme ces résultats devront être transmis à un autre projet pour être affichées, il est nécessaire de convenir d'un protocole que tous les résultats devront respecter. Les détails de ce protocole sont expliqués dans le chapitre 9. Mais en ce qui concerne l'interface, elle doit gérer la fermeture du programme en proposant de sauvegarder les analyses effectuées, si l'utilisateur le désire. Ainsi il faut soit sauvegarder les différents fichiers créés, soit

```java
public class Data {

    // Nom du fichier EDF
    public String fileEDF = null;
    // L'ID du patient
    public String patientID = null;
    // Titre des données
    public String title = null;
    // Série de données
    public float[] data = null;
    // Unité des données
    public String unite = null;
    // Fréquence d'échantillonnage
    public int freq = 0;
    /* -- Interval -- */
    // Echantillon de départ
    public int echDep = 0;
    // Nombre d'échantillon
    public int nbEch = 0;

    /** Creates a new instance of Data */
    public Data() {
    }
    // Initialisation de la longueur de la série de données
    public void initDataLength(int length){
        this.data = new float[length];
    }
}
```

FIG. 7.7 – Code correspondant à la classe *Data*

tous les supprimer en s'assurant qu'il ne reste aucun fichier de cette analyse, ceci pour ne pas encombrer la machine de l'utilisateur inutilement.

7.1.10 Résultat

Après avoir réalisé chaque partie décrite précédemment, il suffit de les placer de manière adéquate sur la fenêtre principale, en ajoutant une gestion de l'activité des buttons. Ainsi tous les buttons sont désactivés au départ du programme, et ils sont activés au fur et à mesure de l'exécution. De ce fait, il n'est pas possible de provoquer une erreur à cause d'une fonction qui ne devrait pas être activée, à un certain moment. Malgré cette gestion, l'utilisateur peut ne pas comprendre comment s'utilise le programme. C'est pour cela, qu'il est très utile d'ajouter un champ « Conseils », qui aide l'utilisateur pas à pas. De plus, un manuel d'utilisation est disponible en annexe A. Tout ceci pour obtenir une interface graphique lisible et facile d'utilisation. La figure 7.8 montre la fenêtre principale du programme en cours d'utilisation.

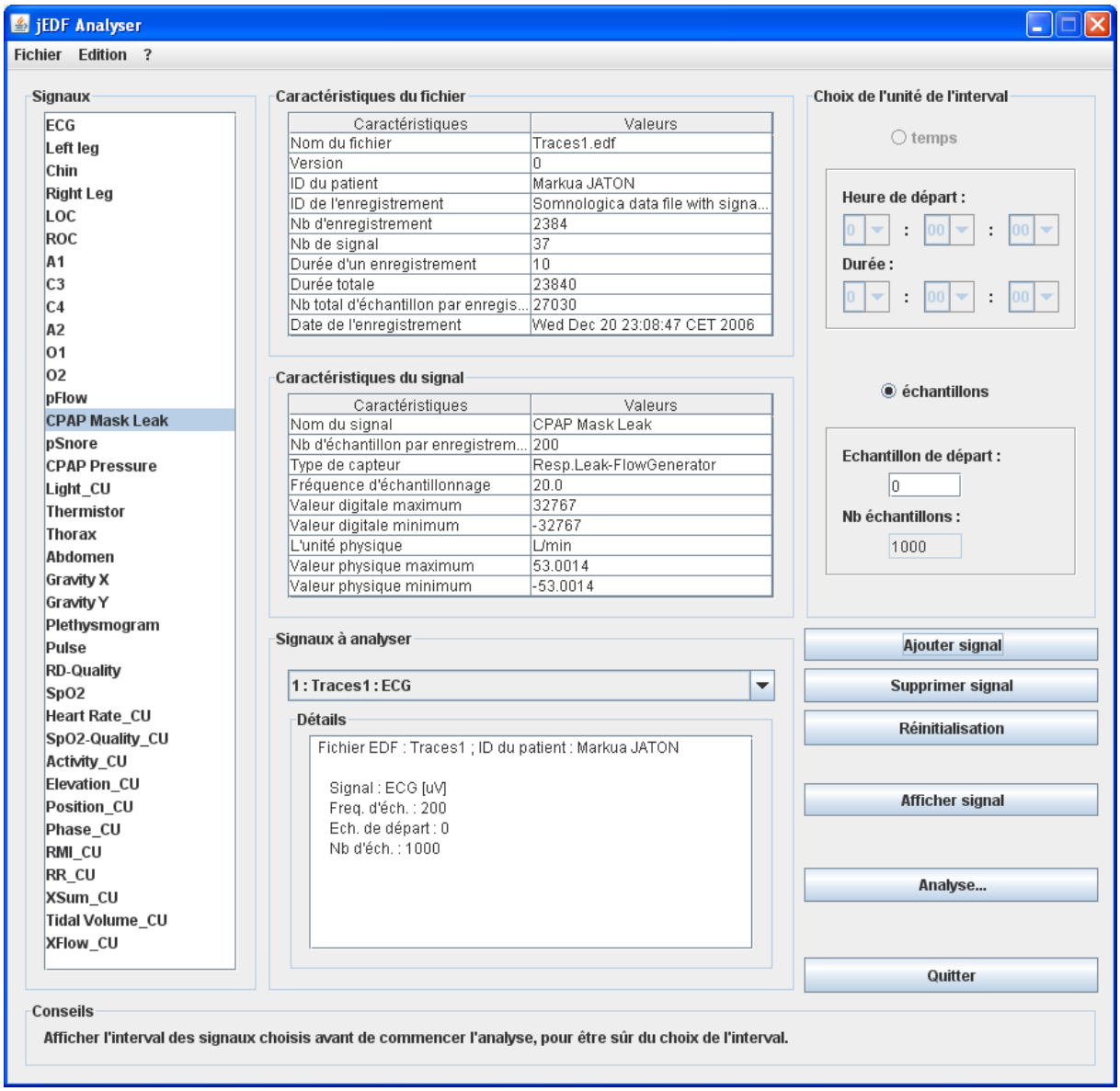


FIG. 7.8 – Fenêtre principale du programme, en cours d'utilisation

7.2 Analyse

Il est question d'appliquer différentes algorithmes à des intervalles de signaux, en permettant à l'utilisateur de choisir l'analyse et l'étendue de cette dernière. La manière la plus facile et claire trouvée pour réaliser cette fonction, est une fenêtre sous forme de console. Cette dernière possède un écran où s'affichent les résultats des analyses de chaque signal et une ligne où l'utilisateur peut entrer une commande (correspondant à une analyse). Mais aussi une liste de toutes les analyses possibles avec leur explication.

Dans la suite de ce chapitre, une analyse, une commande et une fonction se réfèreront à la même chose, à savoir un plugin. Pour plus d'informations voir le chapitre 8.1.

7.2.1 Liste des commandes

Comme expliqué précédemment, il est nécessaire d'afficher la liste des commandes disponibles ainsi que leur explication. Comme d'habitude, il n'est pas possible d'afficher toutes les explications à l'écran, donc on en affiche juste une à la fois, en affichant les explications de la commande sélectionnée dans la liste. Pour donner un exemple, la figure 7.9 montre un avec trois signaux.

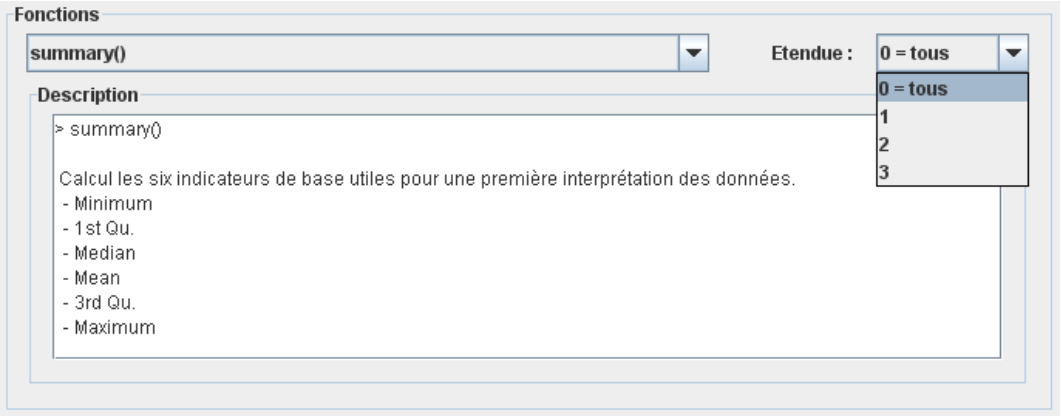


FIG. 7.9 – Exemple de la liste des commandes

Certaines commandes peuvent avoir des paramètres qui doivent être fournis par l'utilisateur, ceci par l'intermédiaire de la ligne de commande. Lors de l'exécution d'une de ces commandes, le programme demande au plugin combien de paramètres il attend, et vérifie si l'utilisateur en a entré le bon nombre, et si ce sont bien des valeurs de type entier. Puis les paramètres sont transmis au plugin pour l'analyse, sous forme d'un tableau d'entiers.

La liste déroulante « Etendue » comme on peut le voir à la figure 7.9, permet de choisir si l'analyse s'applique à tous les signaux, ou un en particulier. Pour le moment, il n'est pas possible de choisir plusieurs signaux à analyser, sans prendre tous ceux de la liste, mais ceci sera détaillé dans le chapitre 11.

De plus, il est bien sûr évident que chaque commande doit être unique pour ne pas poser de problème d'exécution. Donc, il est possible d'ajouter des plugins à ceux déjà existant, mais il ne doit pas avoir un nom de commande qui existe déjà. Les détails concernant l'ajout d'un plugin (ou d'une commande) seront expliqués dans le chapitre 8.

7.2.2 Ligne de commande

Cette ligne de commande permet à l'utilisateur de choisir l'analyse voulue, et de transmettre les paramètres voulus si nécessaire. Pour simplifier le travail de l'utilisateur, la liste des analyses existantes est reliée à la ligne de commande, ainsi la fonction choisie dans la liste s'affiche automatiquement dans le champ d'exécution. L'utilisateur doit encore remplacer le nom de chaque paramètre par la valeur choisie

et enfin l'exécuter. Comme exemple, la figure 7.10 montre la console avant l'exécution d'une analyse de corrélation. Le paramètre donné par l'utilisateur définit la méthode utilisée pour calculer la corrélation, dans ce cas il s'agit de la méthode de « Pearson ».

7.2.3 Ecran

Cet écran est simplement une zone de texte qui n'est éditable que par le programme, ainsi pour l'utilisateur il s'agit juste d'un écran où s'affiche les résultats des analyses qu'il a choisi. Pour présenter ces résultats de manière plus présentable, il est utile d'y afficher un petit en-tête avec les signaux traités et leurs informations. La figure 7.10 montre un exemple avec trois signaux, avant l'exécution de l'analyse de corrélation, tandis que la figure 7.11 montre la console après cette exécution.

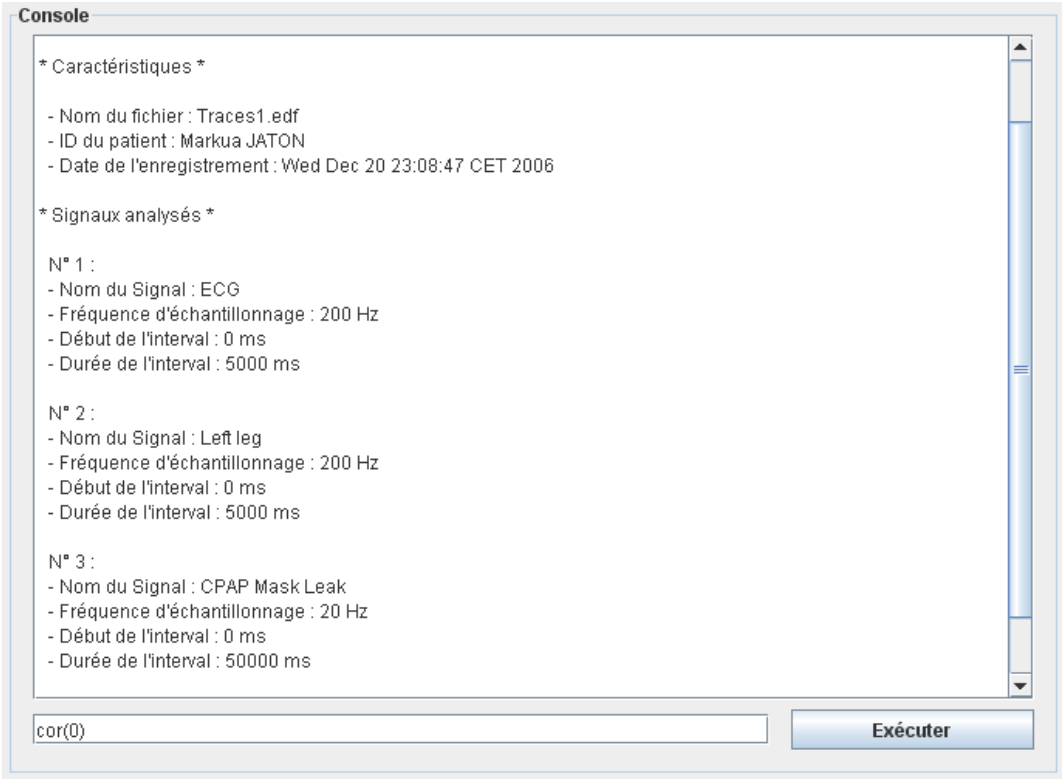

Console
* Caractéristiques *
- Nom du fichier : Traces1.edf
- ID du patient : Markua JATON
- Date de l'enregistrement : Wed Dec 20 23:08:47 CET 2006
* Signaux analysés *
N° 1 :
- Nom du Signal : ECG
- Fréquence d'échantillonnage : 200 Hz
- Début de l'interval : 0 ms
- Durée de l'interval : 5000 ms
N° 2 :
- Nom du Signal : Left leg
- Fréquence d'échantillonnage : 200 Hz
- Début de l'interval : 0 ms
- Durée de l'interval : 5000 ms
N° 3 :
- Nom du Signal : CPAP Mask Leak
- Fréquence d'échantillonnage : 20 Hz
- Début de l'interval : 0 ms
- Durée de l'interval : 50000 ms
cor(0)
Exécuter

FIG. 7.10 – Exemple de la console avant l'analyse de corrélation

Les résultats de l'analyse ne sont pas facile à afficher propremment dans une simple zone de texte, mais heureusement que les résultats finaux ne seront pas enregistrés sous ce format. Les détails à ce sujet se trouvent dans la chapitre 9.

7.2.4 Résultat

Comme pour la fenêtre principale, la figure 7.12 montre la fenêtre analyse en cours exécution.

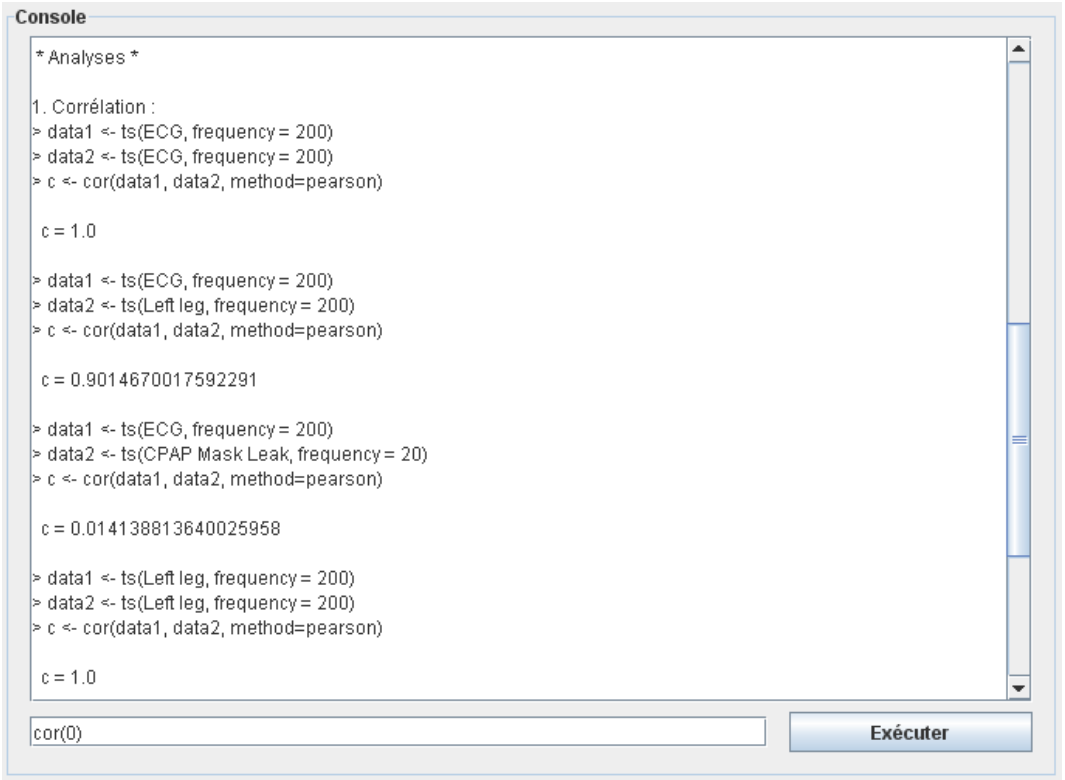


FIG. 7.11 – Exemple de la console après l'analyse de corrélation

L'interface réalisée est agréable et facile d'utilisation, elle implémente toutes les fonctions décrites dans les chapitres précédents. Pour mieux comprendre ce principe d'analyses (plugins), le chapitre suivant explique leurs différentes fonctions en détail.

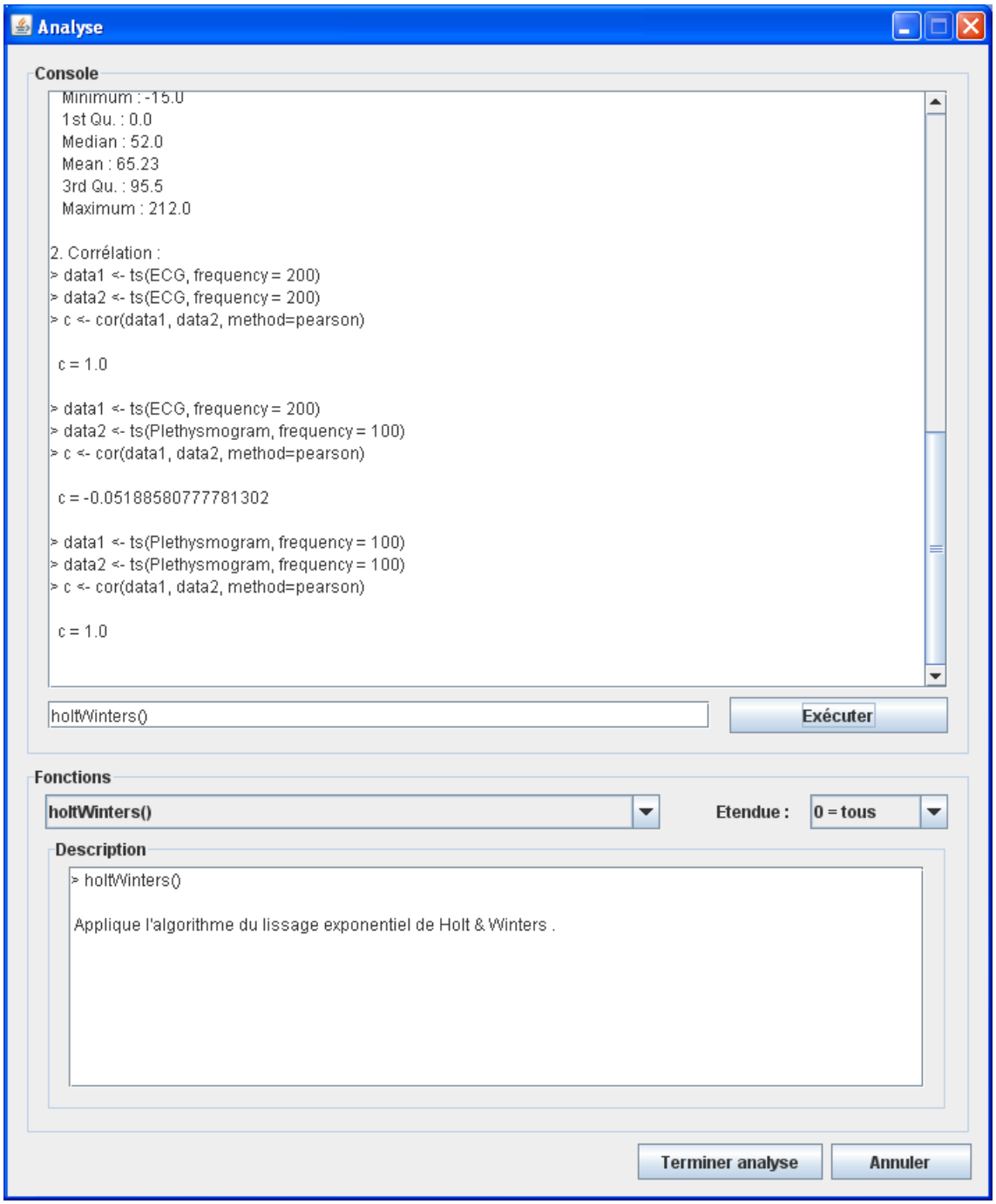


FIG. 7.12 – Fenêtre d'analyse en cours d'exécution

Chapitre 8

Analyses (Plugins)

Ce chapitre contient l'explication des différentes fonctions que chaque plugin devra implémenter (interface *Plugin*), comme mentionné dans le chapitre 6. De plus, une liste expliquative des différents plugins (commandes) réalisés et un explicatif pour la création d'un nouveau plugin sont détaillés. Pour ne pas répéter chaque fois les mêmes explications, le code d'un seul plugin est analysé. Celui d'ARIMA est choisi, car c'est le plugin le plus complet parmi ceux créés.

8.1 Plugin

Une fois l'interface graphique réalisée, les fonctions nécessaires pour la bonne excéution des plugins sont mieux connues et peuvent être listées :

- Une pour appliquer l'analyse, en retournant les séries de données modifiées ou pas.
- Une pour créer l'annotation correspondante à l'analyse (fichier XML, voir chapitre 9).
- Une pour donner le nombre d'annotations créées.
- Une pour décrire l'utilisation de la commande correspondante.
- Une pour décrire l'analyse effectuée et les résultats trouvés.
- Une pour donner le nombre d'arguments nécessaire à cette commande.

Pour mieux comprendre ces fonctions, la figure 8.1 affiche le code de l'interface « Plugin ».

Comme on peut le voir, la fonction qui applique l'analyse (*getResult()*) reçoit, à part le tableau de série de données, toute une série de paramètres dont la liste des arguments et la variable de type *AnalyseFile*, qui seront expliqués ultérieurement. La variable de type *Rconnection* permet de se connecter au serveur Rserve et d'envoyer des commandes R, comme expliqué dans le chapitre 5. Tandis que le compteur donne le numéro de l'analyse en cours, pour différencier les analyses lors de la lecture des résultats.

Une autre fonction intéressante est *paramRes()*, qui crée une annotation pour le fichier XML (voir chapitre 9) pour chaque signal analysé. Les paramètres de cette

```
public interface Plugin {

    // Retourne les séries de données résultantes de l'analyse
    public Data[] getResult(Data[] data, int[] arguments, AnalyseFile af,
                            Rconnection c, int comptAnalyse);
    // Crée l'annotation correspondante à l'analye
    public Resultat paramRes(Resultat r, int comptAnn);
    // Retourne le nombre d'annotations créées
    public int getNbAnn();
    // Retourne une description de la fonction du plugin
    public String getDescription();
    // Retourne une explication de l'analyse
    public String getExplication();
    // Retourne le nombre d'arguments nécessaire
    public int getNbArg();

}
```

FIG. 8.1 – Code correspondant à l'interface *Plugin*

fonction sont la variable correspondante au fichier XML et le compteur des annotations. Comme il est possible qu'une seule analyse crée plusieurs annotations, les plugins doivent retourner le compteur d'annotations. Ceci est réalisé par le fonction *getNbAnn()*, qui retourne le nombre d'annotations créées par le plugin. Ainsi la gestion des annotations est bouclée.

Pour les autres fonctions, il n'y a rien de très spécial, elles retournent des informations correspondantes à l'analyse, pour plus de détail, voir partie 8.3.

8.2 Plugins réalisés

Le tableau 8.1 contient toutes les explications des plugins réalisés lors de ce projet de diplôme, pour prouver que l'analyse de fichier EDF est possible.

Plugin	Commande	Paramètre	Explication	Intérêt
Affichage	`affichage()`	Aucun	Affiche les intervalles fournis.	Premier plugin testant la sauvegarde d'un graphique transmis par Rserve.
ARIMA	`arima(ar, i, ma, sar, si, sma, p)`	Degré de chaque partie du modèle ARIMA et de son effet saisonnier, ainsi que la période de ce dernier.	Applique le modèle arima définit par les paramètres choisis par l'utilisateur, aux intervalles fournis.	Premier plugin vérifiant si l'application d'un modèle statistique et la récupération du résultat sont possibles.
Correlation	`cor(method)`	Méthode utilisée pour calculer la corrélation (0 = pearson, 1 = kendall, 2 = spearman)	Calcule la corrélation entre tous les intervalles fournis.	Premier plugin appliquant une analyse sur deux intervalles simultanément.
HoltWinters	`holtWinter()`	Aucun	Applique l'algorithme du lissage exponentiel de Holt & Winters.	Plugin testant la récupération d'un autre type de résultat.
Lissage	`lissage(n)`	Diviseur du nombre d'échantillons de l'intervalle	Lisse les intervalles fournis pour diminuer leur nombre d'échantillons.	Premier plugin n'utilisant pas le serveur Rserve, mais uniquement les fonctions de Java.
Summary	`summary()`	Aucun	Calcul les six indicateurs de base, à savoir le minimum, le 1er quantile, la médiane, la moyenne, le 3ème quantile, et le maximum.	Plugin testant la récupérations d'un autre type de résultat.
DetectAbaiss	`detectAbaiss(x)`	Seuil de détection de l'abaissement en pourcent.	Détecte les abaissements des intervalles transmis, supérieur à x%. Ce plugin est spécifiquement réservé au signal « Plethysmogram ».	Analyse demandé par le Dr Heinzer travaillant au centre du sommeil du CHUV.

Tab. 8.1 – Explications des plugins réalisés lors de ce projet de diplôme

8.3 Code ARIMA

Le code du plugin ARIMA est présenté dans l'ordre identique à celui de son exécution. Ainsi on commence par la description du plugin, à savoir l'explication de la commande et de ses paramètres. Cette description est affichée en dessous de la console dans la partie « Analyse » de l'interface graphique (voir partie 7.2.1). La figure 8.2 montre la méthode retournant cette description.

```
// Description
public String getDescription(){
    return "> arima(data, order=c(ar,i,ma), seasonal=list(order=c(sar,si,sma),period=p))\n\n " +
           "  ar : ordre de la partie auto-regressive du modèle ARIMA.\n" +
           "  i : ordre de la partie différencielle du modèle ARIMA.\n" +
           "  ma : ordre de la partie moyenne mobile du modèle ARIMA.\n" +
           "  sar : ordre de la partie auto-regressive de l'effet saisonnier.\n" +
           "  si : ordre de la partie différencielle de l'effet saisonnier.\n" +
           "  sma : ordre de la partie moyenne mobile de l'effet saisonnier.\n" +
           "  p : période de l'effet saisonnier.\n\n" +
           "Applique le modèle ARIMA définit sur les données fournies,\n " +
           "puis prédis le dernier cinquième des données pour comparer.";
}
```

FIG. 8.2 – Méthode *getDescription()* du plugin ARIMA

L'exécution continue avec l'analyse proproprement dite, en faisant appel à la méthode *getResult()* du plugin. Cette fonction applique le modèle ARIMA choisi par l'utilisateur et sauvegarde la prédiction calculée (1/5 de la longueur de l'intervalle) dans un fichier PNG. Les figures 8.3 et 8.4 montrent la méthode getResult du plugin ARIMA.

Comme on peut le constater, cette méthode est le centre du plugin car elle applique même la préparation de l'explication. Cette étape consiste à préparer un explicatif, avec les résultats trouvés (valeurs, tableau, graphique, etc) pour la console et le fichier HTML (voir chapitre 9). Il est bien sûr évident que chaque résultat est transmis de manière différente, un fichier HTML accepte les tableaux et les images, alors que la console n'accepte que du texte. La figure 8.5 montre cette partie avec l'exemple du plugin ARIMA.

On constate que la console utilise la fonction *getExplication()* pour acquérir les explications, alors que le fichier HTML (variable *AnalyseFile af*) est directement édité depuis le plugin. De plus, la méthode *affResult()* retourne, dans le cas du plugin ARIMA, un tableau des coefficients du modèle statistique réalisé. Mais cette dernière méthode n'est pas obligatoirement présente dans tous les plugins, c'est pour cela qu'elle n'est pas exposée.

Tout ceci ne servirait à rien si on ne crée pas une annotation (voir chapitre 9) correspondante à cette analyse, c'est le rôle de la méthode *paramRes()*, dont la figure 8.6 montre l'exemple du plugin ARIMA.

```
// Applique le modèle ARIMA définit par l'utilisateur sur les intervals transmis
public Data[] getResult (Data[] data, int[] arguments, AnalyseFile analyseF,
                         Rconnection c, int comptAnalyse){
    d = data;
    af = analyseF;
    comptAnalyse = comptAnalyse;
    arg = arguments;
    p = new Data();

    // Titre de l'analyse
    af.writeSubsubtitle(comptAnalyse + ". Arima :");
    console += comptAnalyse + ". Arima : \n";

    try {
        // Parcours tous les intervals fournis
        for(int i=0;i<d.length;i++){
            if(d[i]!=null){
                // Castage des tableaux de donnée
                temp = new double[(d[i].data.length)];
                nbPredict = d[i].data.length/5;
                tempBis = new double[d[i].data.length-nbPredict];
                for(int j=0;j<d[i].data.length;j++){
                    temp[j] = (double)d[i].data[j];
                }
                for(int j=0;j<d[i].data.length-nbPredict;j++){
                    tempBis[j] = (double)d[i].data[j];
                }
                // Analyse
                c.assign("data",temp);
                c.voidEval("data <- ts(data, frequency = " + d[i].freq + ")");
                c.assign("databis",tempBis);
                c.voidEval("databis <- ts(databis, frequency = " + d[i].freq + ")");
                c.voidEval("y <- arima(databis, order = c(" + arg[0] + "," + arg[1] + "," + arg[2] + "), " +
                           "seasonal=list(order=c(" + arg[3] + "," + arg[4] + "," + arg[5] + "), period=" +
                           arg[6] + "))");
                c.voidEval("p <- predict(y, " + nbPredict + ")");
                c.voidEval("png(file=\"Arima.png\")");
                c.voidEval("plot(data, xlab = \"Time[s]\")");
                c.voidEval("lines(p$pred, col='red')");
                // Récupération de la prédiciton
                predict = c.eval("p$pred").asDoubleArray();
                c.voidEval("dev.off()");
```

FIG. 8.3 – Méthode *getResult()* du plugin ARIMA (partie 1)

8.4 Ajout d'un plugin

Il est bien sûr indispensable de pouvoir ajouter des plugins à cette application, ceci pour la rendre évolutive. Pour ceci, il faut créer une nouvelle classe Java, dans le répertoire « Plugins » du projet, qui sera le plugin en lui-même, et d'ajouter deux ligne de la classe *Commande*. Mais voici tout de suite les détails de ce dernier point.

8.4.1 Classe *Commande*

La classe *Commande* gère la liaison entre les commandes données par l'utilisateur et les plugins. La première chose à faire pour ajouter un nouveau plugin, c'est de compléter la liste des commandes existantes, comme le montre le code de la figure 8.7.

Ensuite, il faut relier cette commande au nouveau plugin créé (nouvelle classe), en ajoutant une possibilité au moment du choix du plugin à exécuter. Ceci est montré

```
                // Récupération des coefficients calculés
                coeff = c.eval("y[[1]]").asDoubleArray();
                sigma = c.eval("y[[2]]").asDouble();
                likelihood = c.eval("y[[5]]").asDouble();
                aic = c.eval("y[[6]]").asDouble();
                // Création d'une Data pour la prédiction
                p.initDataLength(predict.length);
                p.title = d[i].title + "_Arima_" + comptAnalyse;
                p.freq = d[i].freq;
                p.unite = d[i].unite;
                for (int k=0;k<predict.length;k++){
                    p.data[k] = (float)predict[k];
                }

                // Affichage de la prédiction du modèle ARIMA grâce à jfreeChart
                final TimeSeriesPlot ts = new TimeSeriesPlot(null, p.title, p, "Prédiction_Arima");
                ts.pack();
                ts.setVisible(true);

                // Copie du fichier PNG généré
                path = new String(af.getPath().replaceAll("Analyses.html","src"));
                File f = new File("./Arima.png");
                new File(path).mkdir();
                imageName = "Arima_" + comptAnalyse + "_" + i + ".png";
                image = new File(path + "/" + imageName);
                try {
                    this.copyFile(f, image);
                } catch (Exception e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
                f.delete();

                // Préparation des explications
                prepareExplication(d[i]);
            }
        }
    } catch (RSrvException e) {
        JOptionPane.showMessageDialog(null, "Cette fonction ne peut pas être appliquée à ces données, " +
                "ou avec ce paramètre !", "Erreur", JOptionPane.WARNING_MESSAGE, null);
    }

    return d;
}
```

FIG. 8.4 – Méthode *getResult()* du plugin ARIMA (partie 2)

par la figure 8.8.

Après avoir ajouté ces deux lignes, notre plugin peut être exécuté en le sélectionnant dans la liste des analyses. Mais pour qu'il s'exécute correctement, il faut avant tout qu'il implémente les différentes fonctions de l'interface *Plugin*.

L'ajout d'un plugin n'est pas automatique, il nécessite une nouvelle compilation. Mais comme ce projet n'est encore qu'un prototype et qu'il sera modifié de nombreuses fois, je ne pense pas que ce soit une priorité.

8.4.2 Codage du plugin

Il est fortement conseillé de copier le code d'un plugin déjà existant et d'effacer le code contenu dans toutes les fonctions indispensables. Ainsi le nouveau plugin est

```
// Prépare le texte explicatif de l'analyse
public void prepareExplication(Data d){
    // Explication pour le fichier HTML
    explication = explication(d, false);
    af.writeText(explication);
    // Explication pour la console
    console += explication(d, true);
    console += affResult();
}
// Retourne une explication de l'analyse effectuée et du résultat
public String explication(Data d, boolean console){
    String temp = "> data <- ts(" + d.title + ", frequency = " + d.freq + ")\n" +
            "> plot(data, xlab = \"Time[s]\")\n" +
            "> y <- arima(databis, order = c(" + arg[0] + "," + arg[1] + "," + arg[2] + "), " +
            "seasonal=list(order=c(" + arg[3] + "," + arg[4] + "," + arg[5] + "), period=" +
            arg[6] + "))\n" +
            "> p <- predict(y, " + nbPredict + ")\n" +
            "> lines(p$pred, col='red')\n\n" +
            " - Fichier enregistré sous : ";
    if(console){
        temp += image.getPath() + "\n\n";
    } else {
        temp += "./src/" + imageName + "\n\n";
    }
    temp += "> y \nCoefficients : \nSigma est estimé à " + sigma + " ; log likelihood = " +
            likelihood + " ; aic = " + aic + "\n\n";
    return temp;
}
// Retourne un texte explicatif de l'analyse
public String getExplication(){
    return console;
}
```

FIG. 8.5 – Méthode *prepareExplication()* et explication du plugin ARIMA

prêt à être compléter. Mise à part la fonction principale *getResult()* qui est conséquente, le reste des fonctions est facile à compléter. Les fonctions *getDesription()* et *getExplication()* demandent respectivement une petite explication de la commande et une explication du résultat. De plus *getNbArg()* et *getNbAnn()* permettent de préciser respectivement le nombre d'arguments nécessaires à l'exécution de l'analyse, et le nombre d'annotations créées durant cette dernière. Pour finir, il reste la fonction pour paramétrer les annotations créées (*paramRes()*), selon les résultats obtenus par l'analyse.

Il est clair que toutes ces fonctions dépendent plus ou moins fortement de *getResult()*. C'est pour cela qu'il est recommendé de la compléter en premier et ensuite de remplir les autres. Ainsi, les étapes à réaliser pour créer un nouveau plugin ont été présentées, et nous ne rentrerons pas plus en détail dans les différentes possibilités, vu qu'elles sont presque illimitées.

```
// Retourne les annotations réalisées (variable fichier XML)
public Resultat paramRes(Resultat res, int compt){
    String comment = "Analyse liee avec le(s) signal(aux) suivant(s) : ";
    int cpt = 0;
    for(int i=0;i<d.length;i++){
        if(d[i]!=null){
            cpt++;
            comment += d[i].title + " - ";
        }
    }
    if(cpt==1){
        comment = "";
    }
    // Chemin d'accès du fichier html, depuis le fichier xml
    char[] c = af.getPath().toCharArray();
    int j = 0;
    while(c[j]!='\\'){j++;}
    j++;
    while(c[j]!='\\'){j++;}
    j++;
    while(c[j]!='\\'){j++;}
    int length = c.length-(j+1);
    String path = new String("");
    for(int i=0;i<length;i++){
        path += c[j+1+i];
    }
    // Parcours tous les intervals
    for(int i=0;i<d.length;i++){
        if(d[i]!=null){
            res.initAn(compt);
            // Nom du signal
            res.setAnNomSignal(d[i].title, compt);
            // Titre de l'annotation
            res.setAnTitre("ARIMA_" + (compt+1), compt);
            // Chemin du fichier HTML
            res.setFichierHTML(path, compt);
            // Commentaire
            res.setAnComments(comment, compt);
            // Convertion de l'interval en millisecondes
            int debutMilli = (int)((double)d[i].echDep * (1000/d[i].freq));
            int dureeMilli = (int)((double)d[i].nbEch * (1000/d[i].freq));
            // Interval du signal
            res.setAnDebutMillis(debutMilli, compt);
            res.setAnDureeMillis(dureeMilli, compt);
            compt++;
            comptAnn++;
        }
    }
    return res;
}
```

FIG. 8.6 – Méthode *paramRes()* du plugin ARIMA

```
// Liste des commandes reconnues
public String[] listCmd = {"affichage()",
                           "arima(ar, i, ma, sar, si, sma, p)",
                           "cor(method)",
                           "holtWinters()",
                           "lissage(n)",
                           "summary()",
                           "detectAbaiss(x)",
                           "detectAbaissII(x)"
                           // "Xxx()"    // Ajout d'une commande
                          };
```

FIG. 8.7 – Liste à compléter pour l'ajout d'une commande

```
// Recherche le plugin correspond et retourne sa description
public String findCmd(int index){
    switch(index){
        case 0 : plugin = new Affich(); break;
        case 1 : plugin = new Arima(); break;
        case 2 : plugin = new Correlation(); break;
        case 3 : plugin = new HoltWinters(); break;
        case 4 : plugin = new Lissage(); break;
        case 5 : plugin = new Summary(); break;
        case 6 : plugin = new DetectAbaiss(); break;
        case 7 : plugin = new DetectAbaissII(); break;
        // case 8 : plugin = new Xxx(); break;       // Ajout d'une commande
        default : plugin = null;
    }
    if (plugin != null){
        description = plugin.getDescription();
        return description;
    }
    return null;

}
```

FIG. 8.8 – Mise à jour du choix du plugin à exécuter

Chapitre 9

Transmission du résultat

Ce projet fait partie du projet IMINET (Intelligent Medical Information Network, voir [1]), qui a pour but de créer une cabinet médical virtuel. Pour la partie analyse et affichage de données EDF, deux projets différents ont été dédiés. De ce fait, il est impératif que ces deux projets soyent compatible (transmission des annotations de l'analyse à l'affichage), mais heureusement les deux projets ont été choisis cette année (affichage EDF : M. Gavin ; analyse EDF : moi-même). Ceci permet de connaître les besoins des deux côtés et de fixer un protocole fiable et sûr.

9.1 Discussion

Lors de notre première discussion concernant la transmission du résultat, nous avons décidé de placer toutes les informations concernant l'analyse dans un fichier XML, en donnant une liste de fichiers joints qui contiennent les différents résultats des analyses. Ainsi nous avons établit le format du fichier XML avec un exemple.

Mais après divers tests, ce format pose quelques problèmes pour le projet de M. Gavin, car il n'est pas facile de gérer la diversité possible des fichiers joints. Alors nous avons dû retravailler ce point, pour que le fichier XML ne contienne qu'un fichier joint qui présentera la série d'analyse ainsi que les résultats obtenus. Le type de fichier idéal pour cela est un fichier HTML, car il peut contenir du texte, des tableaux et des images, et la présentation est nettement mieux que dans une zone de texte (console). Nous avons réécris le format du fichier XML en y acceptant qu'un seul fichier HTML.

9.2 Fichier XML

Comme mentionné précédemment, le fichier XML doit contenir les diverses informations concernant le fichier EDF traité, et les intervals des signaux analysés. Voici donc une liste de toutes les informations présentes dans ce fichier :

- Nom du fichier EDF
- Date de la capture
- Date de l'analyse
- Liste des annotations

Une annotation correspond à une analyse effectuée sur un interval d'un signal. Ainsi lors d'une analyse sur n signaux, il y aura n annotations créées qui se réfèreront toutes au même fichier HTML. Voici la liste des informations nécessaires à la création d'une annotation :

- Nom du signal
- Titre pour l'annotation
- Début de l'interval en ms
- Durée de l'interval en ms
- Commentaire qui doit contenir la liste des signaux qui ont subis la même analyse
- Nom du fichier HTML

Pour mieux visualiser ce format, la figure 9.1 montre le schéma correspondant, de plus un exemple d'un fichier XML est affiché par la figure 9.2.

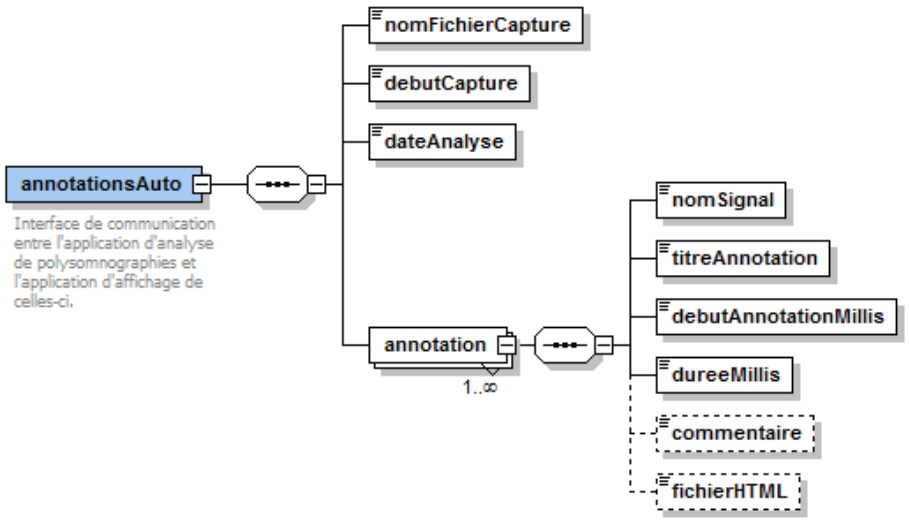


Fig. 9.1 – Schéma du format XML définit

9.3 Fichier HTML

Le fichier HTML contient exactement les mêmes informations que la console de la partie « Analyse » de l'interface graphique à la seule différence que la présentation est améliorée grâce à la possibilité d'insérer des tableaux, des images et des listes. Les figures 9.4 et 9.5 montrent le même exemple utilisé pour présenter la console dans la partie 7.2.3, mais cette fois sous la forme du fichier HTML.

Le résultat obtenu est déjà mieux que celui fournit par la console, mais il est toujours possible de faire mieux, ce point sera traité plus en détail dans le chapitre 11.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<annotationsAuto xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:noNamespaceSchemaLocation="Interconnexion.xsd">
    <nomFichierCapture>String</nomFichierCapture>
    <debutCapture>2001-12-17T09:30:47.0Z</debutCapture>
    <dateAnalyse>2001-12-17T09:30:47.0Z</dateAnalyse>
    <annotation>
        <nomSignal>String</nomSignal>
        <titreAnnotation>String</titreAnnotation>
        <debutAnnotationMillis>0</debutAnnotationMillis>
        <dureeMillis>0</dureeMillis>
        <commentaire>String</commentaire>
        <fichierHTML>String</fichierHTML>
    </annotation>
    <annotation>
        <nomSignal>String</nomSignal>
        <titreAnnotation>String</titreAnnotation>
        <debutAnnotationMillis>0</debutAnnotationMillis>
        <dureeMillis>0</dureeMillis>
        <commentaire>String</commentaire>
        <fichierHTML>String</fichierHTML>
    </annotation>
</annotationsAuto>
```

FIG. 9.2 – Exemple d'un fichier XML

9.4 Arborescence des fichiers

Le dernier point convenu avec M. Gavin, est la forme de l'arborescence des résultats. En effet, il n'est pas possible de placer tous les fichiers créés dans le même dossier, car ceci provoquerait des conflits au niveau des noms. Ainsi il est nécessaire de mettre en place une arborescence fixe pour tous les résultats.

Cette arborescence commence dans un dossier « Résultats », qui contient un dossier pour chaque fichier traité, avec comme nom, la date de l'analyse suivi de l'heure et du nom du fichier EDF. Ce dossier contient le fameux fichier XML contenant les annotations des divers intervals de signaux analysés, ainsi que les noms des fichiers HTML correpsondants. Mise à part ce fichier, il y a aussi un dossier pour chaque série d'analyses, portant le numéro de cette dernière. Un dossier comme celui-ci contient un fichier HTML explicatif, avec un autre dossier s'appelant « src » contenant tous les images générées par cette série d'analyses, et qui sont présentées dans un fichier HTML. Pour y voir plus clair parmi toutes ces explications, la figure 9.3 montre un exemple de cette arborescence.

Ce résultat facilite grandement le travail pour le projet d'affichage de M. Gavin, car il suffit d'accéder au fichier XML voulu, qui contient l'emplacement du fichier HTML enregistrés, et enfin créer un lien vers ce fichier HTML.

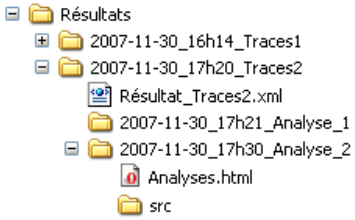


FIG. 9.3 – Exemple de l'arborescence des résultats

* jEDF Analyser *

Caractéristiques

- Nom du fichier : Traces.edf
- ID du patient : Markua JATON
- Date de l'enregistrement : Wed Dec 20 23:08:47 CET 2006

Signal(aux) analysé(s)

N° 1

- Nom du Signal : ECG
- Fréquence d'échantillonnage : 200 Hz
- Début de l'interval : 0 ms
- Durée de l'interval : 5000 ms

N° 2

- Nom du Signal : Left leg
- Fréquence d'échantillonnage : 200 Hz
- Début de l'interval : 0 ms
- Durée de l'interval : 5000 ms

N° 3

- Nom du Signal : CPAP Mask Leak
- Fréquence d'échantillonnage : 20 Hz
- Début de l'interval : 0 ms
- Durée de l'interval : 50000 ms

FIG. 9.4 – Exemple d'un fichier HTML (partie 1)

Analyses

1. Corrélation :

```
> data1 <- ts(ECG, frequency = 200)
> data2 <- ts(ECG, frequency = 200)
> c <- cor(data1, data2, method=pearson)

> data1 <- ts(ECG, frequency = 200)
> data2 <- ts(Left leg, frequency = 200)
> c <- cor(data1, data2, method=pearson)

> data1 <- ts(ECG, frequency = 200)
> data2 <- ts(CPAP Mask Leak, frequency = 20)
> c <- cor(data1, data2, method=pearson)

> data1 <- ts(Left leg, frequency = 200)
> data2 <- ts(Left leg, frequency = 200)
> c <- cor(data1, data2, method=pearson)

> data1 <- ts(Left leg, frequency = 200)
> data2 <- ts(CPAP Mask Leak, frequency = 20)
> c <- cor(data1, data2, method=pearson)

> data1 <- ts(CPAP Mask Leak, frequency = 20)
> data2 <- ts(CPAP Mask Leak, frequency = 20)
> c <- cor(data1, data2, method=pearson)
```

Corrélation	ECG	Left leg	CPAP Mask Leak
ECG	1.0	0.9014670017592291	0.014138813640025958
Left leg	0.9014670017592291	1.0	0.010114791390326842
CPAP Mask Leak	0.014138813640025958	0.010114791390326842	1.0

FIG. 9.5 – Exemple d'un fichier HTML (partie 2)

Chapitre 10

Mise à jour

Ce chapitre contient les différentes améliorations apportées au projet, que ce soit au niveau de la présentation, de l'efficacité du code ou encore l'amélioration graphique. Le but principal est de faciliter et d'optimiser l'utilisation du programme, ceci dans le temps à disposition. Les améliorations qui n'auront pas eu le temps d'être mise en place seront dans le chapitre 11.

De plus, durant ce projet il a été possible de visiter le centre du sommeil du CHUV, grâce à M. Jaton et au Dr Heinzer. Cette visite a permis de mieux connaître les besoins réels demandés à ce genre de programme et les analyses qui intéressent le médecin spécialisé dans le domaine du troubles de sommeil.

10.1 Présentation du code

Vu l'ampleur de ce projet d'un point de vue programmation, il est impératif de finaliser les commentaires, la présentation générale et les nom des variables utilisées, pour que le code puisse être plus facilement réutilisé par la suite. Les détails de cette relecture ne sont pas expliqués, car ils n'apportent pas de nouvelles informations intéressantes.

Ensuite, il est possible d'améliorer légèrement l'efficacité du code, en vérifiant s'il est possible de simplifier certaines parties du programme. Mais après relecture du code aucunes modifications importantes n'a été réalisées, car ce programme est codé de manière simple, en essayant de le rendre le plus clair possible.

10.2 Améliorations graphiques

En ce qui concerne les améliorations graphiques, le principal changement est la mise en place d'icônes sur les boutons de la page principale, pour rendre l'interface légèrement plus agréable. La figure 10.1 montre le résultat de cet ajout.

De plus, un menu « A propos » a été ajouté, présentant succintement le programme jEDF Analyser, avec l'auteur, la version, l'année de création, et une description du logiciel. La figure 10.2 montre le résultat de ce menu.

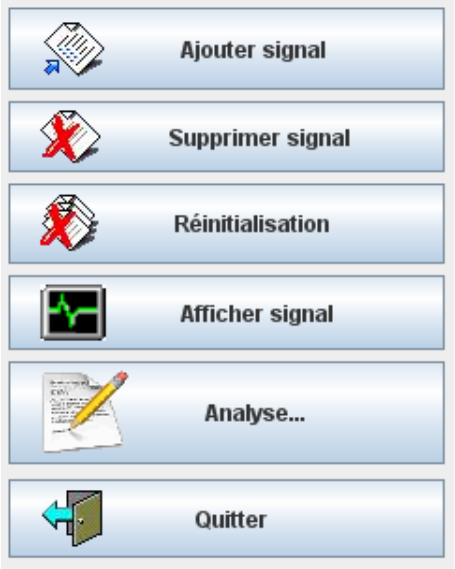


FIG. 10.1 – Ajout des icônes sur les buttons

A propos
jEDF Analyser
Auteur : Julien Boutillier
Création : 2007
Version : 1.0
Description : jEDF Analyser est un programme pour analyser les données d'un fichier EDF, en leurs appliquant divers algorithmes. Il permet aussi d'utiliser des fonctions du logiciel R, pour pouvoir accéder aux nombreux algorithmes déjà existants.

FIG. 10.2 – Fenêtre « A propos »

Pour finir, un icône pour chaque fenêtre du programme a été ajouté, pour donner une touche personnelle à l'application. Le résultat n'est pas présenté ici, mais pour le voir, il suffit d'exécuter le programme.

Pour ce projet, les améliorations graphiques sont terminées, il est bien sûre possible de modifier et d'améliorer encore beaucoup de choses (voir chapitre 11), mais la fin du projet approchant, il est nécessaire de mettre l'axe sur les parties importantes de ce projet (le rapport, et l'analyse de l'abaissement du pouls, voir partie suivante).

10.3 Centre du sommeil du CHUV

Le Dr Heinzer dirige le centre de sommeil du CHUV. Avec son équipe, il utilise des logiciels informatiques spécialisés dans la détection des troubles du sommeil, comme outil d'aide au diagnostique des patients. Malgré ces logiciels, la plus grande partie des analyses est réalisée par les médecins durant, et après l'enregistrement des divers signaux du patient.

10.3.1 Discussion

Durant cette visite le Dr Heinzer nous a montré une analyse qui ne fonctionne pas totalement, pour la détection de l'abaissement du pouls du patient de x% en quelques secondes (x est un paramètre de l'analyse). Il nous a demandé s'il serait possible de mettre en place une analyse similaire, mais plus fiable. Ceci en nous proposant une collaboration possible pour la suite de leurs recherches. Nous avons accepté cette offre en essayant d'avancer un maximum dans cette analyse durant le reste de ce projet.

10.3.2 Analyse de l'abaissement du pouls

Plus en détails, cette analyse doit pouvoir détecter tous les abaissements du pouls du patient supérieur à x%, sur toute la durée du signal. Il faut néanmoins faire attention, car l'amplitude du pouls varie au cours du temps, c'est pour cela qu'il faut vérifier que les abaissements détectés ne soyent pas une diminution de l'amplitude à long terme (+ 30 sec.). La figure 10.3 montre un exemple simple de pouls, avec le résultat que devra fournir l'analyse.

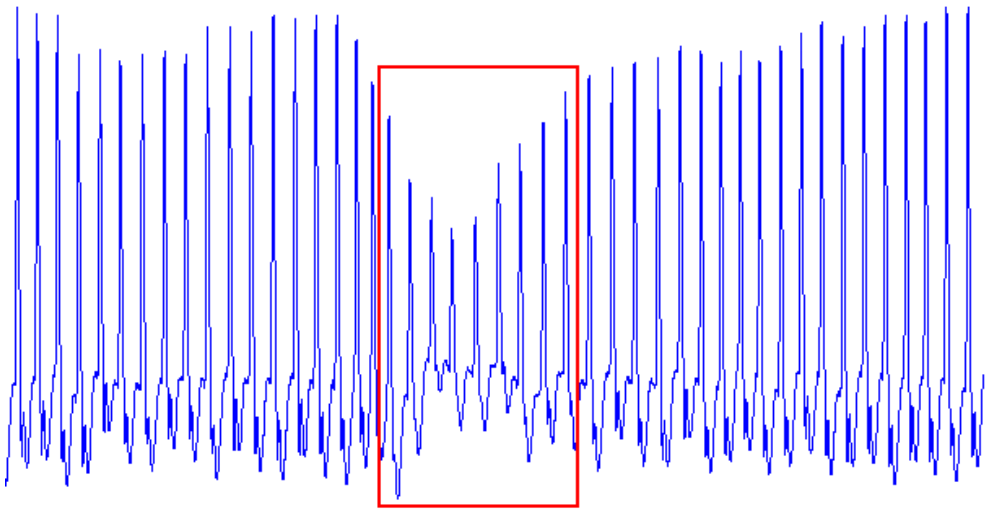


FIG. 10.3 – Exemple d'un signal de pouls avec la détection d'un abaissement

10.3.3 Avancement

Pour commencer cette analyse, il est préférable de disposer des maximas du signal, et non pas de tous ses échantillons. Ceci facilite la détection d'un abaissement, et permet de manipuler moins de données.

Détection des maximas

La solution proposée pour détecter tous les maximas est simple et s'exécute rapidement. Cette solution consiste à créer une fenêtre d'échantillons d'une longueur égale à la période du signal, et de la faire parcourir tout le signal. Pour détecter les maximas, on teste si la valeur centrale de la fenêtre est la plus grande valeur. Si c'est la cas, cette valeur est un maxima, autrement on passe à la valeur suivante. Il faut toutefois faire attention de sauvegarder l'index, pour pouvoir localiser le maxima ultérieurement sur le signal. Lorsqu'une détection est faite, il est possible d'avancer d'une longueur égale à la moitié de la période du signal, car aucune de ces valeurs ne sera plus grande que la maxima détecté. La figure 10.4 en page 56 montre le code correspondant à ces explications.

Détection des abaissements

Une fois les maximas obtenus, il est possible de détecter les abaissements, mais les solutions possibles sont nombreuses et les résultats sont souvent similaires. Pour le moment une seule sera mise en place pour continuer dans la réalisation de cette analyse. Cette solution consiste à prendre une fenêtre de dix maximas et de parcourir tout le tableau. Ainsi à chaque incrément, la moyenne des cinq derniers maximas est comparée à celle des cinq premiers, si la différence est supérieure à x%, un abaissement est noté. Le code correspondant est montré à la figure 10.5. Pour contrôler les résultats, il est préférable de les afficher, pour ceci jfreeChart possède une classe *IntervalMarker*, qui permet d'afficher un marqueur sur un graphique. Ainsi les résultats obtenus peuvent être affichés comme le montre la figure 10.6.

```
    ...

    // Détection des maximas de tout l'interval transmis
    for(int j=middle;j<(d.data.length-middle);j++){
        paramWindow(d.data, j);
        if(testWindow(j)){
            j += middle;
        }
    }

    ...

}
// Paramètre la fenêtre pour la détection des maximas
public void paramWindow(float[] data, int j){
    // Parcours toute la fenêtre en partant du centre
    for(int i=0;i<(middle+1);i++){
        window[middle+i] = data[j+i];
        window[middle-i] = data[j-i];
    }
}
// Test si la valeur central de la fenêtre est un maxima
public boolean testWindow(int j){
    boolean test = true;
    // Parcours toute la fenêtre pour trouver une valeur plus grande que la valeur centrale
    for(int i=0;i<lengthWindow;i++){
        if(window[middle]<window[i]){
            test = false;
        }
    }
    // Si aucune valeur n'est plus grande que la valeur centrale
    if(test){
        maxima[comptMax] = new Maxima(j, window[middle]);
        comptMax++;
    }
    return test;
}
```

FIG. 10.4 – Code correspondant à la détection des maximas

```
    ...

    // Détecte les abaissements du signal
    for(int k=0;k<comptMax-lengthMax;k++){
        initWindowMax(k);
        if(testWindowMax(d.freq)){
            k+=5;
        }
    }

    ...

}
// Initialisation de la fenêtre des maximas pour la détection des abaissements
public boolean initWindowMax(int k){
    if(k<(comptMax-lengthMax)){
        for(int i=0;i<lengthMax;i++){
            windowMax[i] = maxima[k+i];
        }
        return true;
    } else {
        return false;
    }
}
// Test si la différence entre la moyenne des 5 derniers maximas et
// celle des 5 premiers est supérieure au pourcentage donné
public boolean testWindowMax(int freq){
    moy1 = 0;
    moy2 = 0;
    moitie = (lengthMax/2);

    // Calcul la moyenne de la première moitié des maximas
    for(int i=0;i<moitie;i++){
        moy1 += windowMax[i].value;
    }
    moy1 /= moitie;
    // Calcul la moyenne de la deuxième moitié des maximas
    for(int i=moitie;i<lengthMax;i++){
        moy2 += windowMax[i].value;
    }
    moy2 /= moitie;
    // Test la différence entre les deux moyennes
    if((moy1*pourcentage)>(moy2)){
        abaissements[comptAbaiss] = ((windowMax[moitie-1].index)*(1000/freq));
        return true;
    }
    return false;
}
```

FIG. 10.5 – Code correspondant à la détection des abaissements

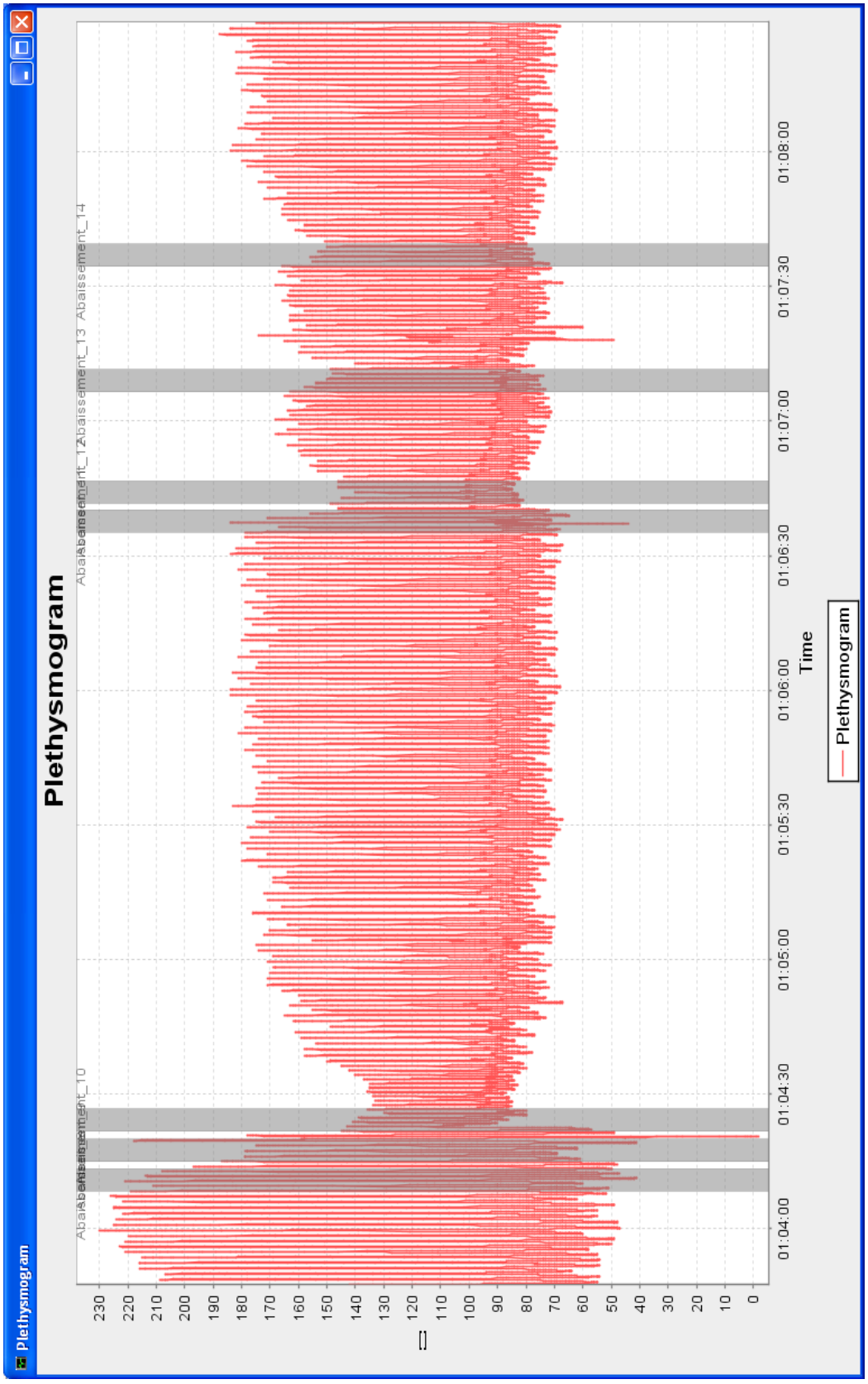


FIG. 10.6 – Exemple du résultat de l'analyse de détection des abaissements

Maintenant que les abaissements sont détectés, il faut vérifier leur durée, et les annoter correctement. Pour être sûr de traiter toutes les possibilités des ces abaissements, il est préférable de les répertorier en relevant chaque cas différent, parmi les trois fichiers EDF à disposition.

Cas 1 La figure 10.7 montre le premier cas d'abaissements détectés, il s'agit d'un abaissement « classique ». A savoir, une diminution des maximas d'au moins 5%, avec le retour à la valeur intiale en environ dix secondes. Ce genre d'abaissement peut varier plus en amplitude ($\sim$ 25%), mais peut aussi durer plus ou moins longtemps (entre 4 et 15 secondes)

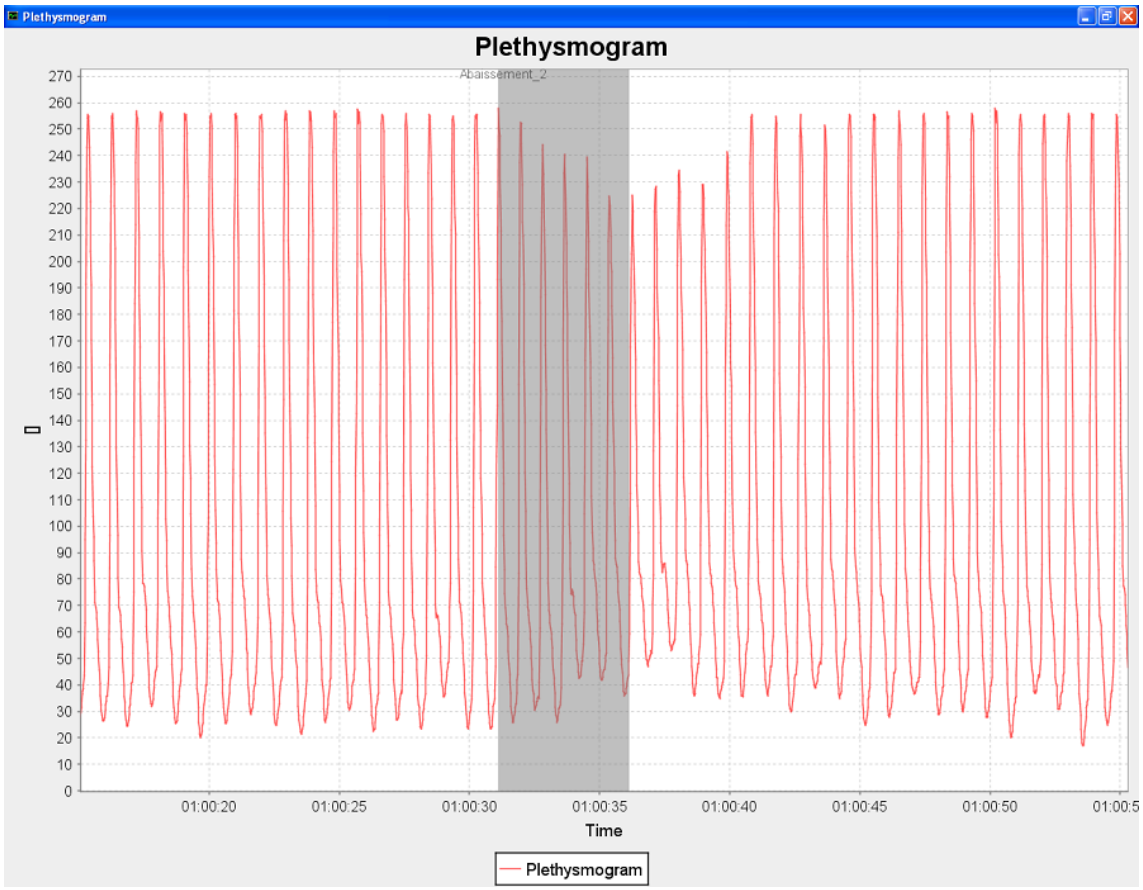


FIG. 10.7 – Exemple du premier cas d'abaissement détecté

Cas 2 La figure 10.8 montre le deuxième cas d'abaissements détectés, il s'agit d'un fort abaissement de l'amplitude d'environ 45%, ceci en à peine 15 secondes. De plus, avec ce genre d'abaissement, l'amplitude du signal ne remonte pas à la valeur initiale, mais à une valeur inférieure d'environ 20%. Ce genre d'abaissement correspond à une chute du pouls du patient, pour en savoir les conséquences, il faudrait l'avis d'un médecin spécialisé.

Cas 3 La figure 10.9 montre le troisième cas d'abaissements détectés, qui dans ce cas ne correspond pas à un abaissement. La présence d'un pic d'amplitude 20%

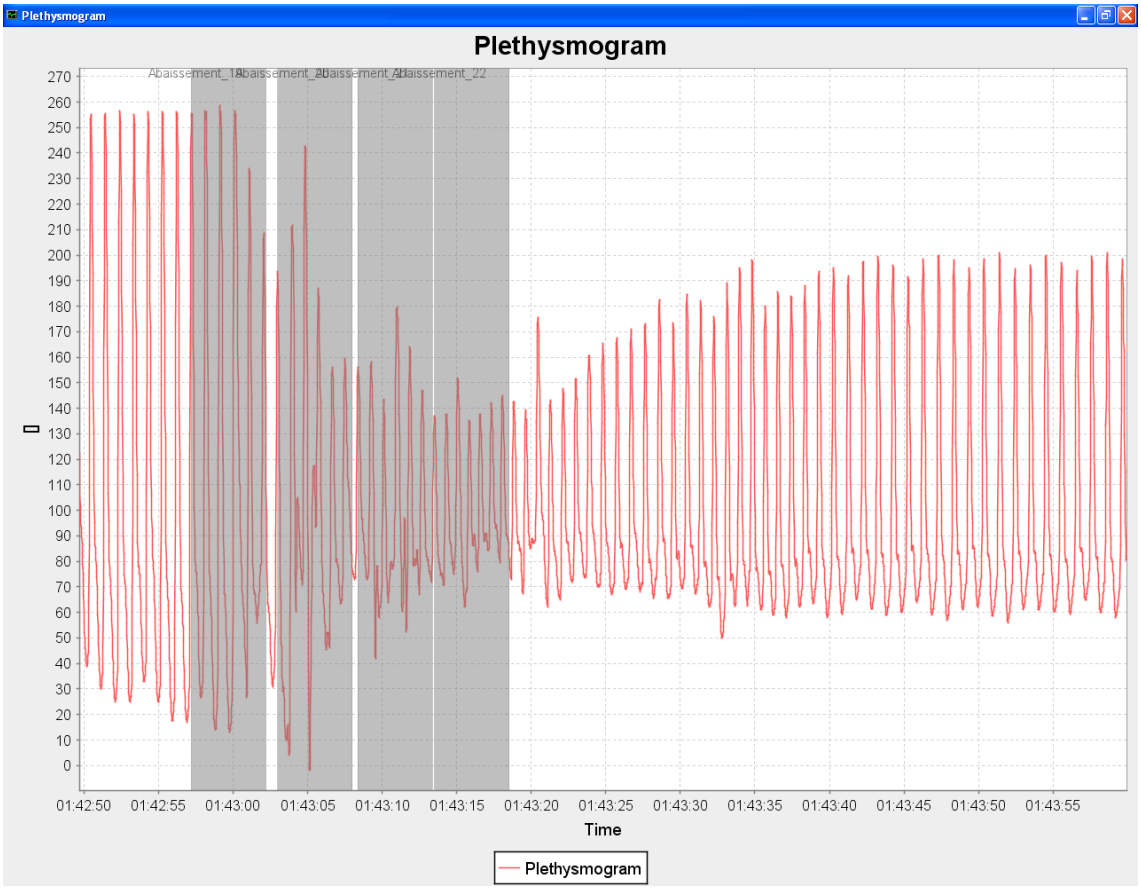


FIG. 10.8 – Exemple du deuxième cas d'abaissement détecté

supérieure à la valeur moyenne des maximas, provoque une détection d'un abaissement, alors qu'il n'y en a pas. Ce genre de problème peut être résolu en filtrant ou lissant le signal préalablement, mais il est difficile de juger la pertinence de nos manipulations, sans l'avis d'un spécialiste.

Cas 4 La figure 10.10 montre le quatrième cas d'abaissements détectés. Ils sont détectés à cause des fluctuations des maximas, car il suffit d'avoir 5 maximas qui fluctuent dans les hautes valeurs, et les 5 suivants qui fluctuent avec des valeurs légèrements inférieures. Ces situations provoquent une détection d'abaissement, alors qu'avec une vision globale, on voit qu'il n'y a pas vraiment un abaissement. Ce problème peut être résolu en augmentant le pourcentage du seuil de détection, mais ce changement pourrait influencer les résultats obtenu dans les cas précédent. Pour avancer dans ce problème, l'aide d'un medecin spécialiste est nécessaire.

Il est vrai que dans les différents cas présentés précédemment, certains ne devraient pas être détectés, mais pour le moment il est préférable de détecté trop d'abaissements, que pas assez. Il est vrai qu'il est plus facile pour les médecins d'annuler un abaissement faux, que de relire tout le signal pour vérifier s'il n'en manque pas.

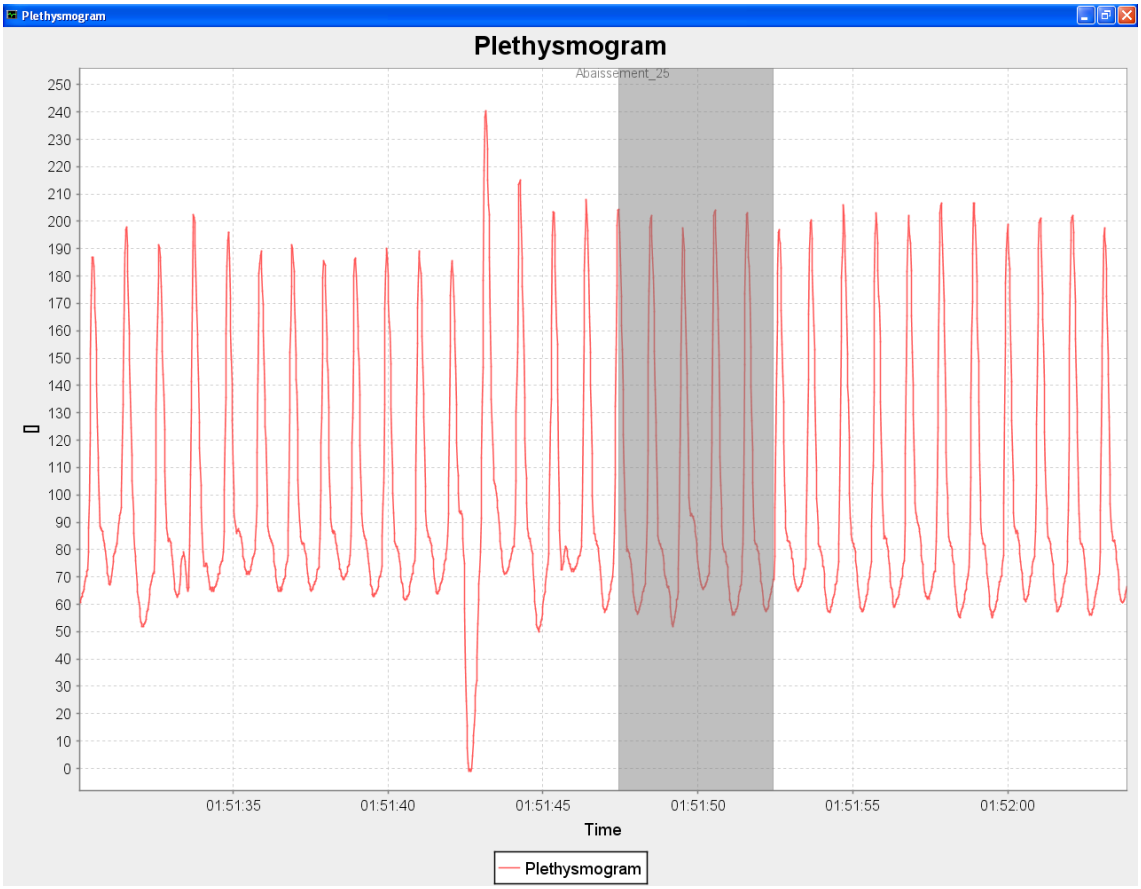


FIG. 10.9 – Exemple du troisième cas d'abaissement détecté

Durée d'un abaissement

Une dernière partie est rajoutée à cette analyse, elle consiste à détecter la durée d'un abaissement et de marquer l'annotation correspondante sur toute cette durée. Si la durée d'un abaissement n'arrive pas à être « mesuré », l'annotation correspondante aura une durée de 5 secondes.

Le principe utilisé pour réaliser ceci et le même que celui expliqué pour la détection des abaissements, mais utilisé à l'envers. En effet, il détecte les augmentations qui suivent les abaissements. L'analyse teste les 30 secondes suivantes et si aucune augmentation n'est détectée pendant ce temps, l'annotation durera 5 secondes, comme mentionné précédemment. Ces cas sont interpréter comme de abaissements du pouls à long terme, ils sont néanmoins indiqués pour que les médecins spécialisés puissent définir si, oui ou non, ces abaissements sont importants. La figure 10.11 montre le code correspondant à cette partie, et la figure 10.12 montre le même exemple présenté la figure 10.6, mais avec l'analyse finale.

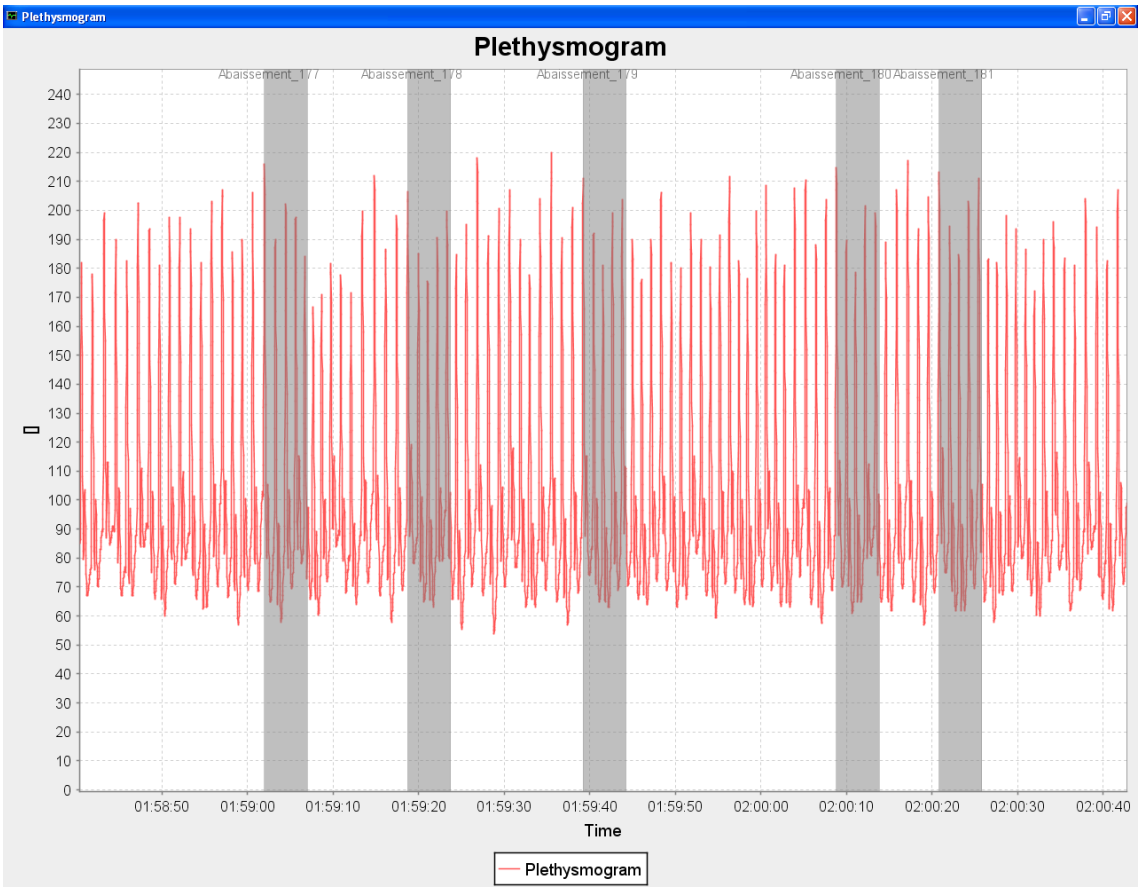


FIG. 10.10 – Exemple du quatrième cas d'abaissement détecté

```
    ...
    // Détecte les abaissements du signal
    for(int k=0;k<comptMax-lengthMax;k++){
        initWindowMax(k);
        if(testWindowMax(d.freq)){
            k+=3;
            // Détecte la fin de l'abaissement
            temp = detectEnd(k, d.freq);
            if(temp!=0){
                k = temp;
            }
            comptAbaiss++;
        }
    }
    ...
}
// Détecte la fin des abaissements
public int detectEnd(int k, int freq){
    int res, index = k;
    moy1 = 0;
    moy2 = 0;
    moitie = (lengthMax/2);

    // Parcours les 30 prochains maximas
    for(int j=0;j<30;j++){
        // Mise à jour de la fenêtre des maximas
        if(!initWindowMax(index)){
            // Fin de l'abaissement pas trouvé
            res = 0;
            finAbaiss[comptAbaiss] = res;
            return 0;
        }
        // Calcul la moyenne de la première moitié des maximas
        for(int i=0;i<moitie;i++){
            moy1 += windowMax[i].value;
        }
        moy1 /= moitie;
        // Calcul la moyenne de la deuxième moitié des maximas
        for(int i=moitie;i<lengthMax;i++){
            moy2 += windowMax[i].value;
        }
        moy2 /= moitie;
        // Test la différence entre les deux moyennes
        if((moy2*pourcentage)>moy1){
            // Fin de l'abaissements détecté
            res = windowMax[lengthMax-1].index;
            finAbaiss[comptAbaiss] = (res*(1000/freq));
            return index+lengthMax-1;
        } else {
           // Maxima suivant
           index++;
        }
    }
    // Fin de l'abaissement pas trouvé
    res = 0;
    finAbaiss[comptAbaiss] = res;
    return 0;
}
```

FIG. 10.11 – Code correspondant à la détection de la durée d'un abaissement

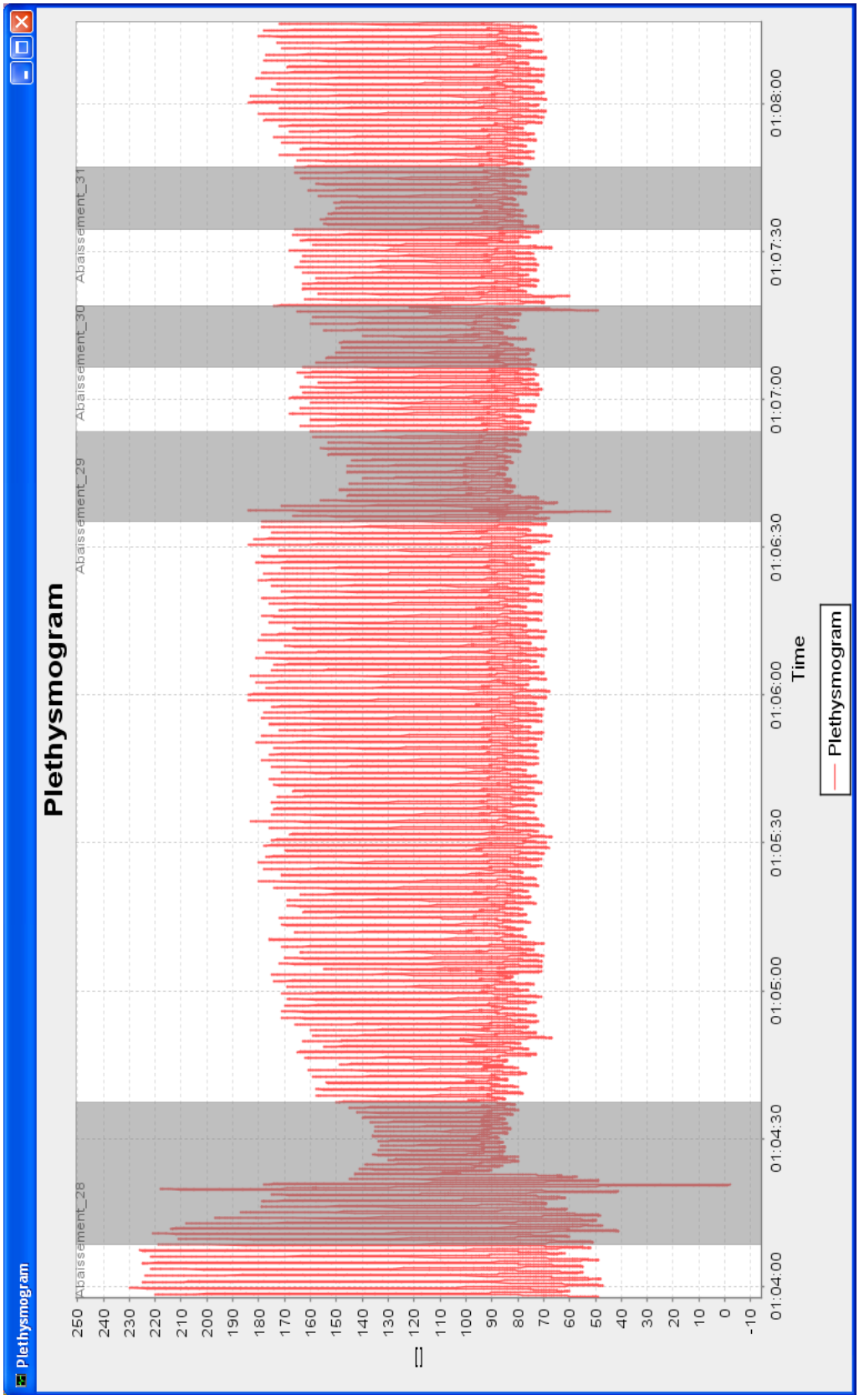


FIG. 10.12 – Exemple du résultat de l'analyse finale

Pour voir le résultat des annotations transmises au projet d'affichage de M. Gavin, des exemples sont disponible en annexe B. Pour améliorer cette analyse, il faudrait une nouvelle discussion avec les medecins du centre du sommeil du CHUV. Comme cette visite n'a pas eu lieu avant la fin du projet, l'analyse a été laissée en l'état actuel.

Chapitre 11

Perspectives

Il est toujours possible d'améliorer un projet, mais il faut savoir s'arréter à temps pour la restitution du dossier. Ce chapitre regroupe les différentes idées d'améliorations qui n'ont pas pues être mise en place durant ce projet.

11.1 Amélioration du fichier HTML

La présentation du fichier HTML peut être améliorée, en changeant la police du texte, le font de la page, etc. Cette amélioration n'est pas spécialement utile, mais on peut imaginer que l'on veuille établir un format précis pour le fichier HTML de résultat de l'analyse. Si c'est le cas, il serait facile de le modifier comme on le désire.

11.2 L'étendue d'une analyse

Pour le moment, comme expliqué dans la partie 7.2.1, il est possible d'appliquer une analyse soit à toutes la liste des signaux à analyser, soit à un seul signal de cette liste. Il serait intéressant de pouvoir choisir plusieurs signaux, sans prendre toute la liste, mais ceci complique légèrement le code de cette partie pour une fonctionnalité mineure. De plus, vu qu'il est déjà possible de choisir les signaux qui seront dans la liste pour l'analyse, il n'est pas vraiment intéressant de pouvoir préciser ce choix.

11.3 Ajout d'analyses

Il serait bien sûr intéressant d'ajouter une multitude d'analyses, mais il faudrait être en mesure de choisir lesquelles sont utiles pour la médecine et lesquelles ne le sont pas. Ceci n'est pas facile à faire, si nous ne sommes pas assistés par un médecin spécialisé.

L'analyse de l'affaissement du pouls se dirige dans cette direction, malheureusement le temps à disposition n'a pas permis de terminer cette analyse à 100%. Mais l'approche débutée dans ce projet est un bon début d'analyse pour toutes personnes voulant la continuer plus en détails.

11.4 Gestion d'analyse multi-fichier EDF

Une autre fonctionnalité à ajouter est la possibilité de réaliser des analyses sur des signaux provenants de fichiers EDF différents. Pour le projet actuel ceci serait réalisable facilement, mais le problème se situe dans la transmission de ces résultats pour le projet d'affichage. Car ce dernier ne peut pas afficher deux signaux de deux fichiers EDF simultanément. Comme ce problème est difficile à résoudre sans devoir changer le protocole de transmission, et poser des problèmes pour les projets de M. Gavin et moi-même, il a été mis de côté.

11.5 Couplage avec un viewer EDF

Cette perspective correspond à un nouveau projet complet à réaliser, car il s'agit de coupler un viewer EDF normal avec ce projet d'analyse. Plus en détail, il faudrait prendre par exemple le viewer EDF de Nizar Kerkeni [7], et lui ajouter la possibilité de choisir l'analyse à effectuer sur l'interval choisi (avec la souris). En effet, ce viewer propose déjà une analyse unique, qui est une FFT. Ainsi si on arrive à isoler l'exécution de cette FFT et à la rediriger vers une liste d'analyses, et ajouter une gestion de cette dernière, on obtient un programme réalisant les mêmes fonctions que le projet actuel, mais avec une meilleure manière de choisir l'interval, une meilleure présentation et une simplicité d'utilisation inégalable. La figure 11.1 montre le viewer EDF de Nizar Kerkeni après une FFT sur un interval choisi.

De plus, il serait intéressant d'ajouter les possibilités du projet de M. Gavin. Ainsi il serait possible d'afficher directement les annotations créées. Avec ce dernier ajout, on obtiendrait un projet complet pour l'analyse et l'affichage des fichiers EDF.

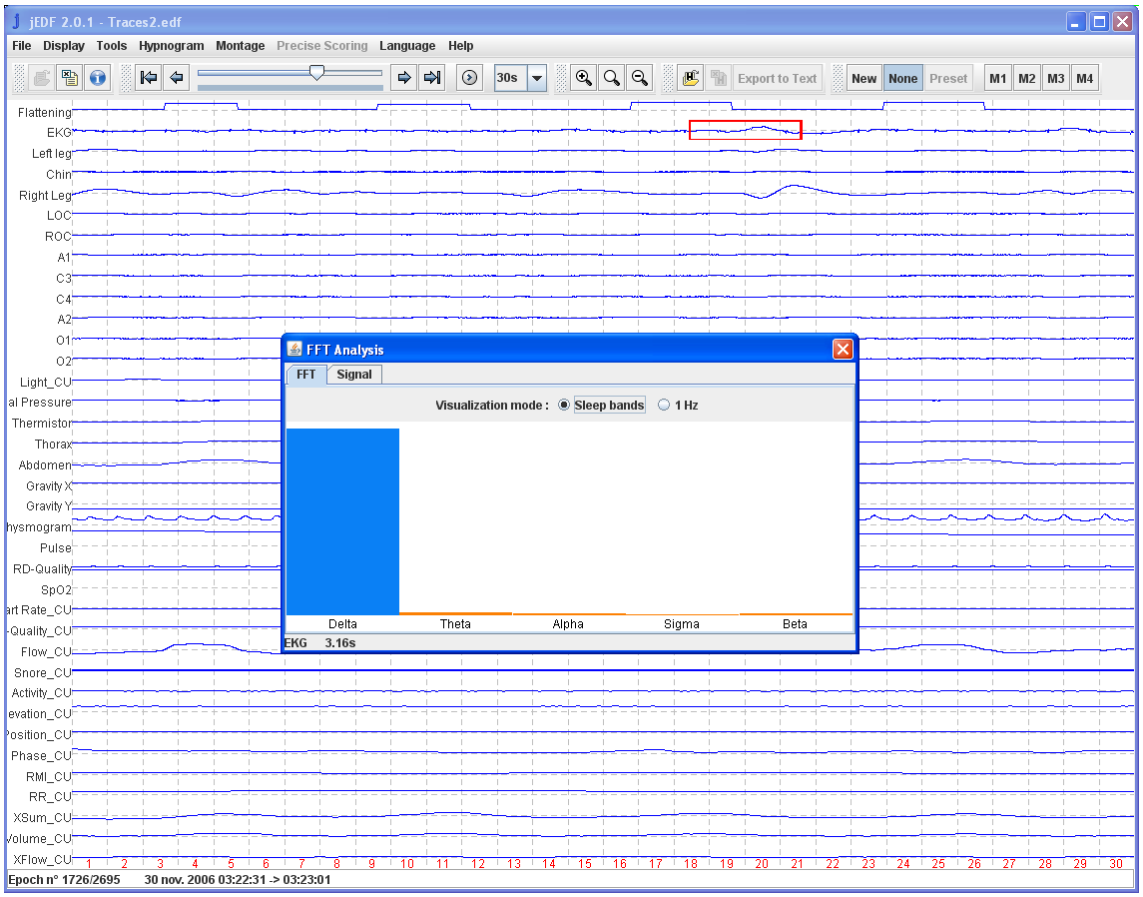


Fig. 11.1 – Viewer EDF de Nizar Kerkeni en cours d'utilisation

Chapitre 12

Conclusion

A la fin de ce projet de diplôme, l'application possède une interface simple et facile à utiliser. Elle est capable d'extraire des données d'un fichier EDF avec leurs caractéristiques, de permettre à l'utilisateur de choisir les intervalles des signaux qu'il veut analyser et de leur appliquer les analyses voulues parmis celles déjà créées. De plus, l'utilisateur peut aussi créer ces propres analyses et les ajouter au programme.

Plus en détail, ces analyse peuvent appliquer toutes sortes d'algorithmes réalisés sous Java, mais aussi faire appel à des fonctions de R, grâce à une connexion au serveur Rserve. Cet atout augmente considérablement les possibilités d'analyses et ainsi la modularité de ce projet.

En ce qui concerne la sauvegarde des résultats des analyses, elle est faite selon le protocole mis en place par M.Gavin et moi-même. Ainsi les annotations résultantes des analyses sont écrites dans un fichier XML, et les coefficients trouvés et autres résultats sont affichés dans un fichier HTML, comme expliqué dans le chapitre 9.

Ce travail a prouvé qu'il est possible d'obtenir les données EDF et de leurs appliquer divers calculs. Il est vrai que la solution proposée ne révolution pas encore l'analyse des données EDF, mais vu qu'il est facile de l'évoluer, tout reste à faire.

Ce projet de diplôme m'a permis de mieux me connaître en ce qui concerne la gestion d'un projet et des délais à respecter. Dans l'ensemble ce projet c'est bien passé, avec ses problèmes à résoudre, ses choix à faire, et ses différents tests à réaliser. De plus, le cahier des charges est respecté, mêmes si de nombreuses améliorations peuvent être ajoutées.

Un point important est de mettre des priorités sur les diverses parties à réalisées, ainsi la fonctionnalité est plus importante que la présentation, par exemple. Ce choix n'est pas toujours évident à faire, surtout s'il y a beaucoup de possibilités.

La possibilité de visiter le centre du sommeil du CHUV a permis une meilleure compréhension des attentes de ce genre de programme. L'analyse demandée par le Dr Heinzer fut très intéressante à mettre en place, mais des améliorations sont toujours envisageables.

Liste des références

[1] Site du projet IMINET (Intelligent Medical Information Network). http://iminet.iict.ch.

[2] Jacques Zuber. Probabilité et statistique. HEIG-VD, 2006.

[3] Stephan Morgenthaler. *Introduction à la statistique*. Presses polytechniques et universitaires romandes, 2001.

[4] Didier Delignières. Séries temporelles - Modèles ARIMA. *Séminaire EA « Sport - Performance - Santé »*, 2000.

[5] Site d'information pour diverses recherches (série temporelle, moindres carrés, etc). http://fr.wikipedia.org.

[6] Site du format EDF et EDF+. http://www.edfplus.info/index.html.

[7] Site personnelle de Nizar Kerkeni. http://www.loria.fr/~kerkeni.

[8] Code source du viewer EDF en Java de Nizar Kerkeni (jEDF).

[9] Robert H. Shunway and David S. Stoffer. *Time series analysis and its applications with R examples*. Springer, 2006.

[10] Bernhard Pfaff. *Analysis of Integrated ans cointegrated time series with R*. Springer, 2006.

[11] Code source du programme de traitement d'image imageJ.

[12] Site d'information sur les séries temporelles. http://lib.stat.cmu.edu/general/tsa2/index.html.

[13] Site d'information sur les séries temporelles et les modèles de régression. http://www.invs.sante.fr/publications/2005/series_temporelles/fichiers/sommaire.html.

[14] Site d'information sur les séries temporelles analysées avec R. http://zoonek2.free.fr/UNIX/48_R_2004/20.html.

[15] Site d'information sur l'interface JRI pour utiliser R sous Java. http://rosuda.org/JRI.

[16] Site d'information sur Rserve, pour utiliser R sous Java. http://rosuda.org/Rserve.

[17] Site officiel du language R. http://www.r-project.org/.

[18] Site d'information sur le language R. http://stat.ethz.ch/R-manual/R-patched/doc/html.

[19] Site d'information sur le language R. http://stat.ethz.ch/R-manual/R-patched/doc/html.

[20] Site du projet jfreeChart, avec API. http://www.jfree.org/jfreechart/index.html.

Liste des tableaux

2.1 Différents termes de la statistique 4

3.1 Principe pour détecter un effet périodique 11
3.2 Calcul de la $1^{ère}$ et $2^{ème}$ différence 12

8.1 Explications des plugins réalisés lors de ce projet de diplôme 40

Table des figures

2.1 Diagramme en points 4
2.2 Diagramme branches-et-feuilles 5
2.3 Histogramme 5
2.4 Boîte à moustache 6
2.5 Diagramme camembert 6
2.6 Graphique en nuage de points 7
2.7 Matrice de nuage de points 8

4.1 Chargement du fichier EDF 19
4.2 Accès aux caractéristiques du fichier EDF 19
4.3 Exemple des caractéristiques d'un fichier EDF 19
4.4 Accès aux caractéristiques du signal et de l'affichage des données du signal . 20
4.5 Lecture du signal 21
4.6 Exemple de caractéristiques d'un signal 21

5.1 Exemple basique des fonctions de Rserve 23

6.1 Illustration de l'architecture du programme 24

7.1 Choix du fichier EDF 27
7.2 Affichage des caractéristique du fichier EDF 28
7.3 Affichage de la liste des signaux et des caractéristiques d'un signal 28
7.4 Choix de l'intervalle 29
7.5 Exemple de la liste des signaux pour l'analyse 30
7.6 Exemple de graphique avec jfreeChart 31
7.7 Code correspondant à la classe *Data* 32
7.8 Fenêtre principale du programme, en cours d'utilisation 33
7.9 Exemple de la liste des commandes 34
7.10 Exemple de la console avant l'analyse de corrélation 35
7.11 Exemple de la console après l'analyse de corrélation 36
7.12 Fenêtre d'analyse en cours d'exécution 37

8.1 Code correspondant à l'interface *Plugin* 39
8.2 Méthode *getDescription()* du plugin ARIMA 41
8.3 Méthode *getResult()* du plugin ARIMA (partie 1) 42
8.4 Méthode *getResult()* du plugin ARIMA (partie 2) 43
8.5 Méthode *prepareExplication()* et explication du plugin ARIMA 44
8.6 Méthode *paramRes()* du plugin ARIMA 45
8.7 Liste à compléter pour l'ajout d'une commande 46
8.8 Mise à jour du choix du plugin à exécuter 46

9.1 Schéma du format XML définit 48
9.2 Exemple d'un fichier XML 49
9.3 Exemple de l'arborescence des résultats 50
9.4 Exemple d'un fichier HTML (partie 1) 50
9.5 Exemple d'un fichier HTML (partie 2) 51

10.1 Ajout des icônes sur les buttons 53
10.2 Fenêtre « A propos » 53
10.3 Exemple d'un signal de pouls avec la détection d'un abaissement 54
10.4 Code correspondant à la détection des maximas 56
10.5 Code correspondant à la détection des abaissements 57
10.6 Exemple du résultat de l'analyse de détection des abaissements 58
10.7 Exemple du premier cas d'abaissement détecté 59
10.8 Exemple du deuxième cas d'abaissement détecté 60
10.9 Exemple du troisième cas d'abaissement détecté 61
10.10Exemple du quatrième cas d'abaissement détecté 62
10.11Code correspondant à la détection de la durée d'un abaissement 63
10.12Exemple du résultat de l'analyse finale 64

11.1 Viewer EDF de Nizar Kerkeni en cours d'utilisation 68

A.1 Page principale à l'ouverture de l'application 75
A.2 Fenêtre pour le choix du fichier EDF 76
A.3 Page principale en cours d'utilisation 77
A.4 Fenêtre d'analyse 78

B.1 Printscreen n°1 79
B.2 Printscreen n°2 80
B.3 Printscreen n°3 81
B.4 Printscreen n°4 82

Annexe A

Mode d'emploi

L'utilisation de cette application n'est pas compliquée, une démonstration d'une utilisation classique va suivre pour prouver ceci. Tout d'abord il suffit de lancer jEDFAnalyser.jar, qui ouvre la page principale avec les champs vides, comme le montre la figure A.1.

Dans cet état, il est possible soit d'ouvrir un fichie EDF, soit de quitter, à l'aide du menu « Fichier ». Après avoir choisi « Ouvrir », la fenêtre affiché à la figure A.2 permet de choisir le fichier EDF voulu. En l'ouvrant cela complète automatiquement les champs de la page principale.

A ce moment, il est possible de choisir les intervals des signaux voulant être analysés, pour ce faire, il faut sélectionner le signal dans la liste, fixer l'interval soit en temps, soit en échantillons et ajouter cet interval dans la liste. Le choix du nombre de signaux appartient à l'utilisateur, mais une limite est fixée à dix signaux (cette limite peut être changée facilement au niveau du code). Pour gérer cette liste, il est possible de supprimer un ou tous les éléments ou d'afficher un interval choisi pour vérifier son choix. La figure A.3 montre la page principale en cours d'utilisation.

Quand le choix des intervals est fini, il faut appuyer sur le bouton « Analyse » pour lancer la fenêtre d'analyse, qui est montré à la figure A.4.

Une fois cette fenêtre ouverte, il faut choisir l'analyse à appliquer parmis la liste à disposition et compléter les paramètres[1] si nécessaire avant de l'exécuter. Après l'exécution de l'analyse et l'affichage du résultat dans la console, il est possible de choisir une autre analyse, de changer de signaux ou encore de changer de fichier. Pour les deux dernières propositions, il faut revenir à la page principale, et recommencer comme expliqué précédemment.

Après avoir terminé les analyses voulues et quitter le programme, les différents résultats sont enregistrés dans un dossier « Résultats », selon l'arborescence expliqué dans la partie 9.4. Ces résultats sont aussi enregistrés lors d'un changement de fichier.

[1] La significations des paramètres est donné dans l'explicatif de la commande, juste en dessous de la liste de analyses.

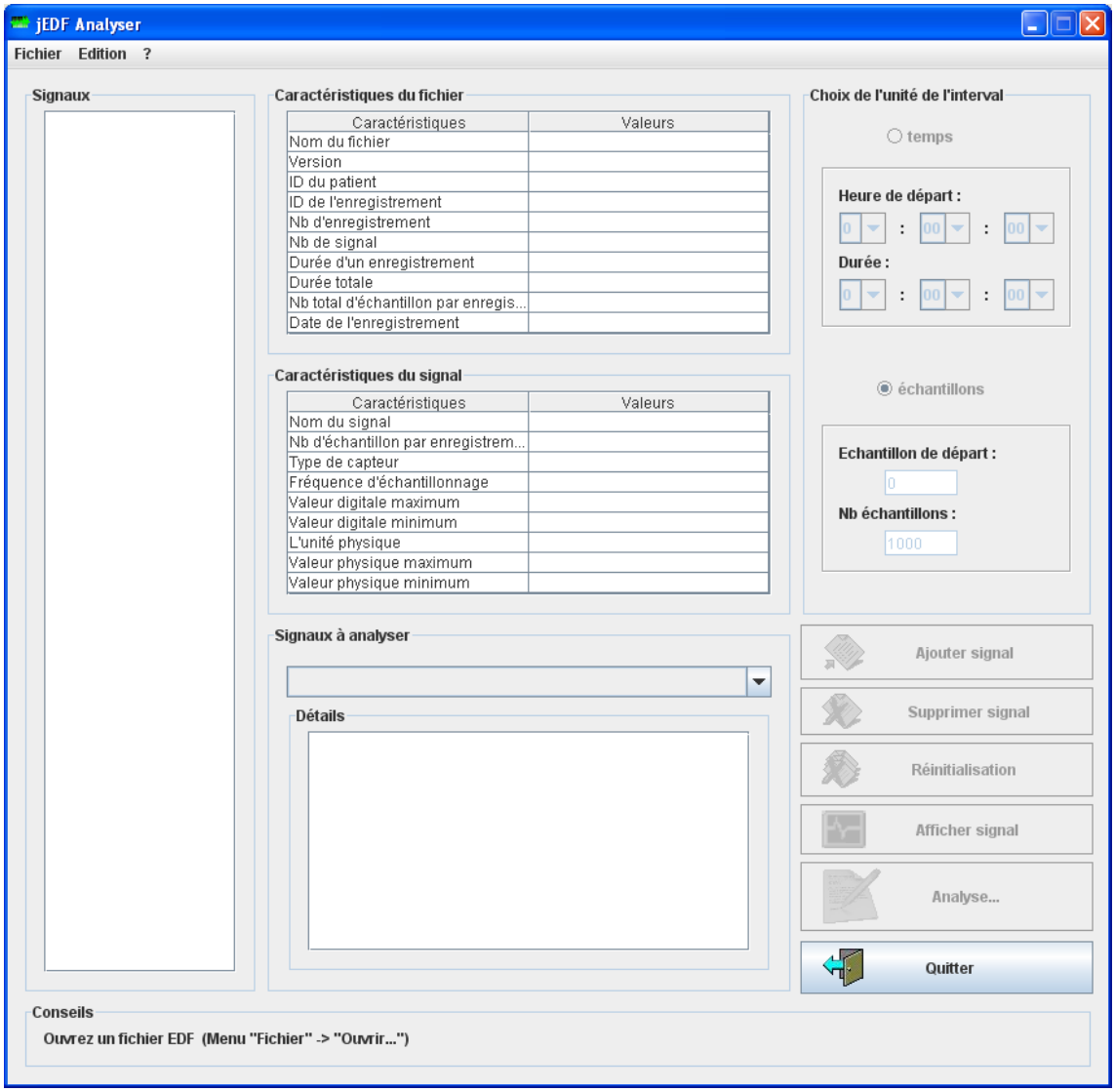


FIG. A.1 – Page principale à l'ouverture de l'application

Voilà tout ce qu'il faut savoir pour utiliser correctement ce programme, et appliquer toutes les analyses voulues. De plus, la partie « Conseils » en bas de la fenêtre principale aide l'utilisateur, en cas de doute sur la marche à suivre.

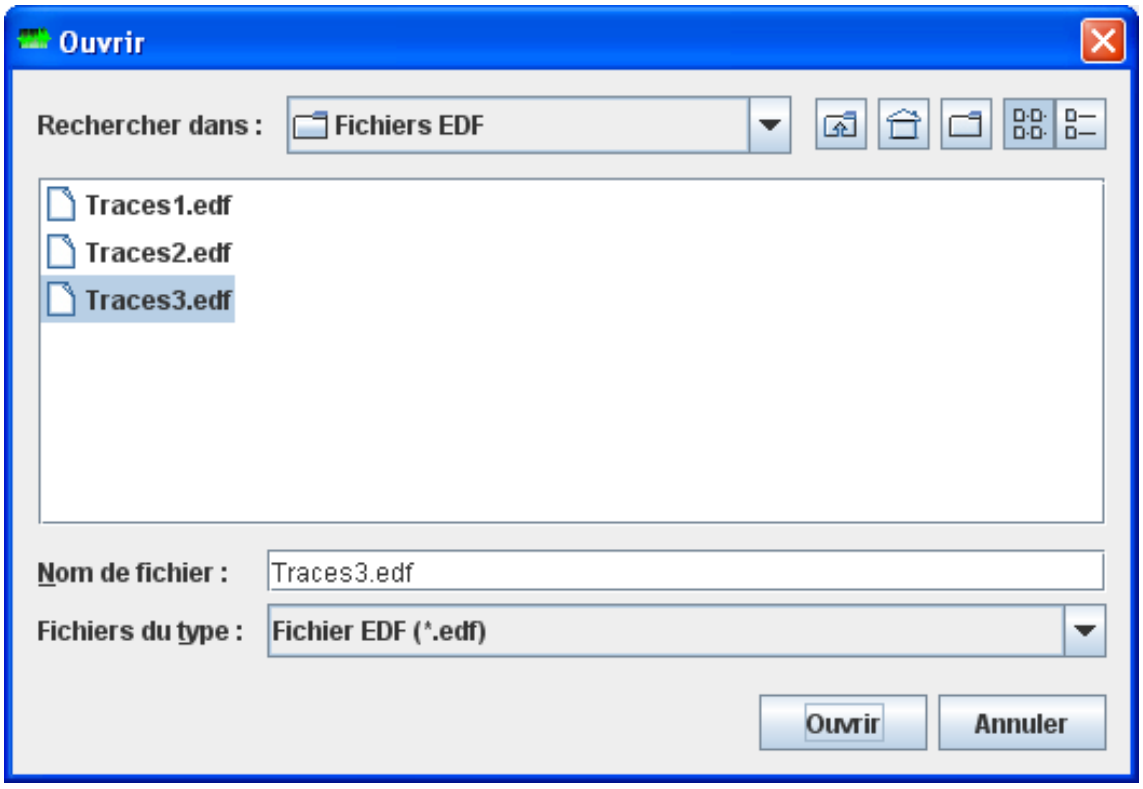


FIG. A.2 – Fenêtre pour le choix du fichier EDF

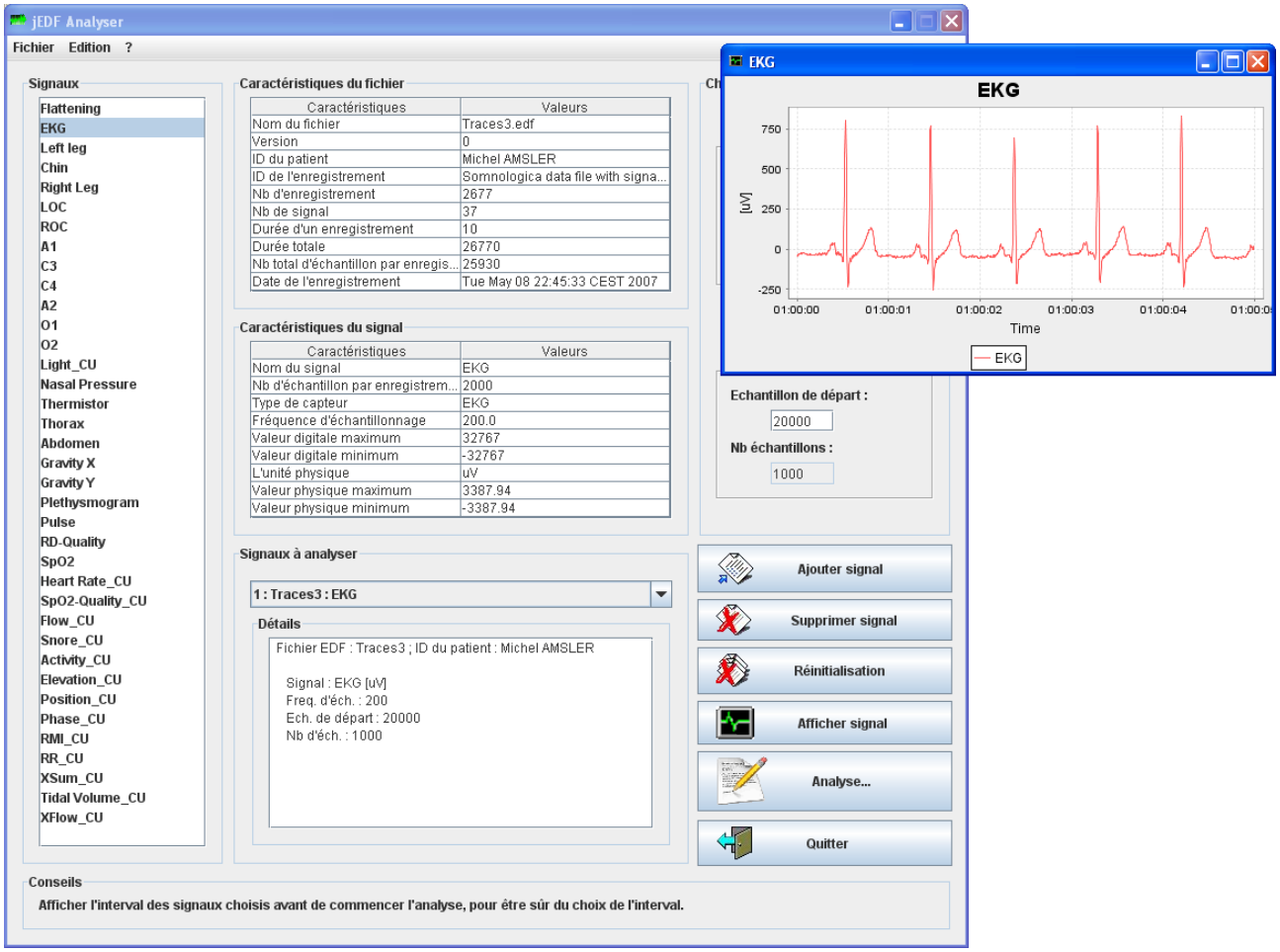


Fig. A.3 – Page principale en cours d'utilisation

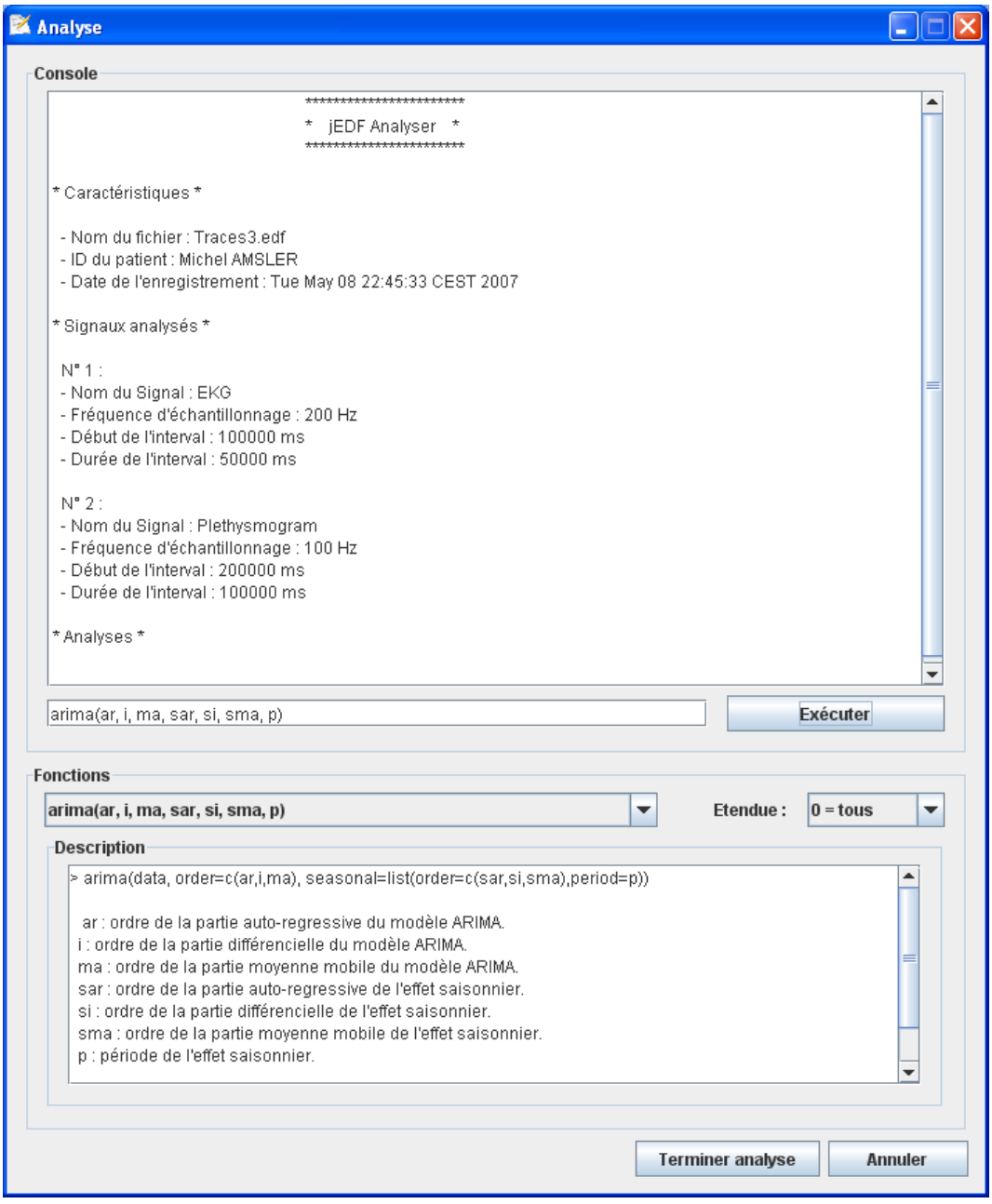

Analyse
Console
****
* jEDF Analyser *
****
* Caractéristiques *
- Nom du fichier : Traces3.edf
- ID du patient : Michel AMSLER
- Date de l'enregistrement : Tue May 08 22:45:33 CEST 2007
* Signaux analysés *
N° 1 :
- Nom du Signal : EKG
- Fréquence d'échantillonnage : 200 Hz
- Début de l'interval : 100000 ms
- Durée de l'interval : 50000 ms
N° 2 :
- Nom du Signal : Plethysmogram
- Fréquence d'échantillonnage : 100 Hz
- Début de l'interval : 200000 ms
- Durée de l'interval : 100000 ms
* Analyses *
arima(ar, i, ma, sar, si, sma, p)
Exécuter
Fonctions
arima(ar, i, ma, sar, si, sma, p)
Etendue :
0 = tous
Description
> arima(data, order=c(ar,i,ma), seasonal=list(order=c(sar,si,sma),period=p))
ar : ordre de la partie auto-regressive du modèle ARIMA.
i : ordre de la partie différencielle du modèle ARIMA.
ma : ordre de la partie moyenne mobile du modèle ARIMA.
sar : ordre de la partie auto-regressive de l'effet saisonnier.
si : ordre de la partie différencielle de l'effet saisonnier.
sma : ordre de la partie moyenne mobile de l'effet saisonnier.
p : période de l'effet saisonnier.
Terminer analyse
Annuler

FIG. A.4 – Fenêtre d'analyse

Annexe B

Résultat du projet de M. Gavin

Les images B.1 à B.4 présentent le résultat obtenu par le projet d'affichage de M. Gavin, avec les annotations transmises depuis mon projet, après l'analyse de détection de l'abaissement du pouls.

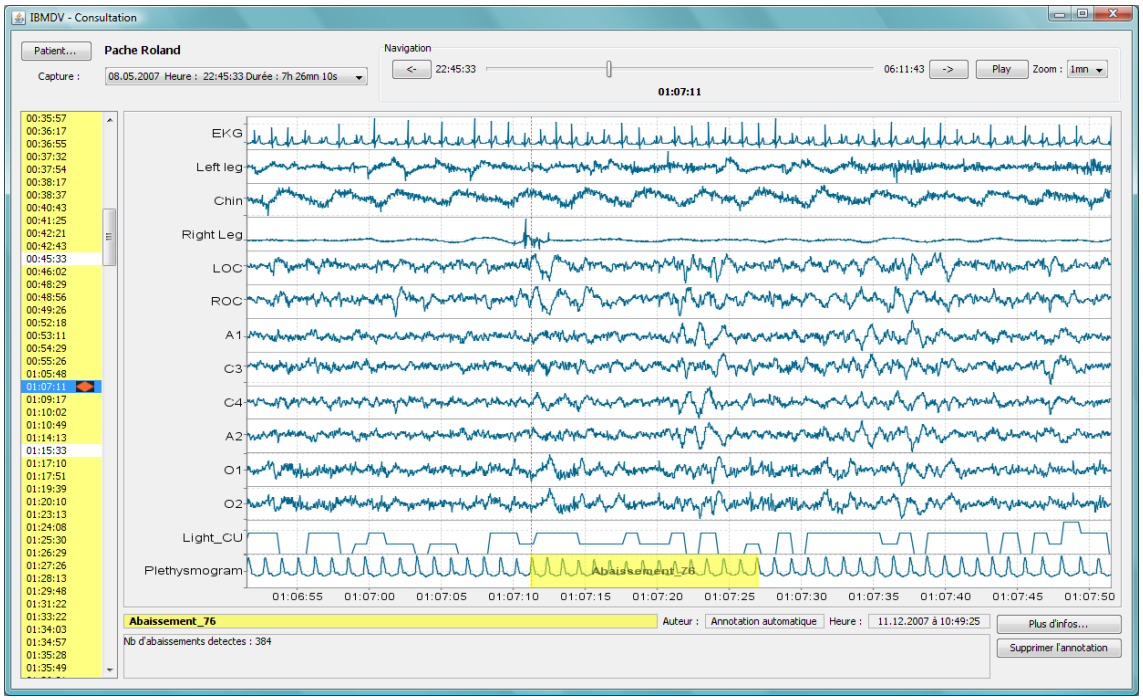


Fig. B.1 – Printscreen n°1

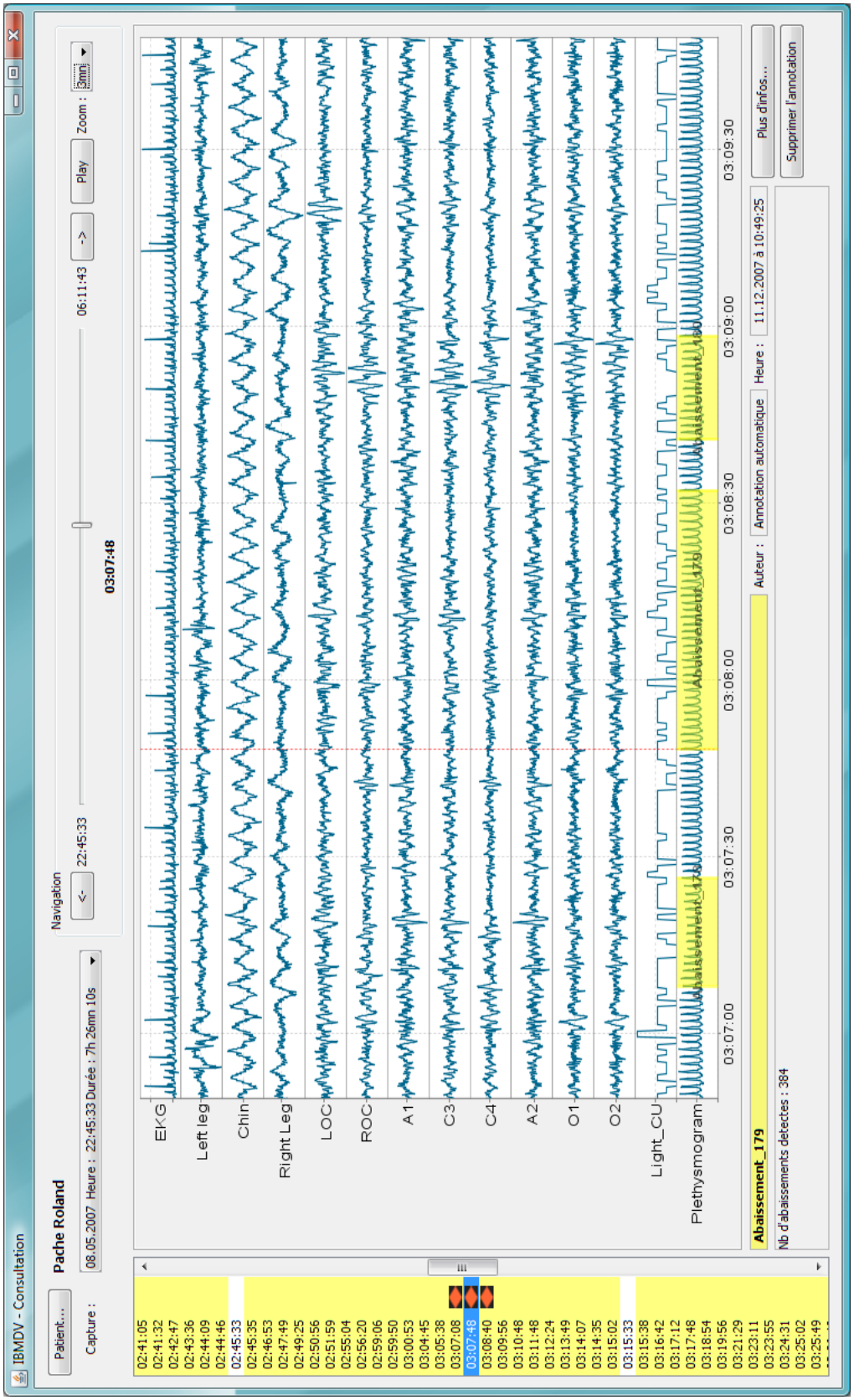


FIG. B.2 – Printscreen n°2

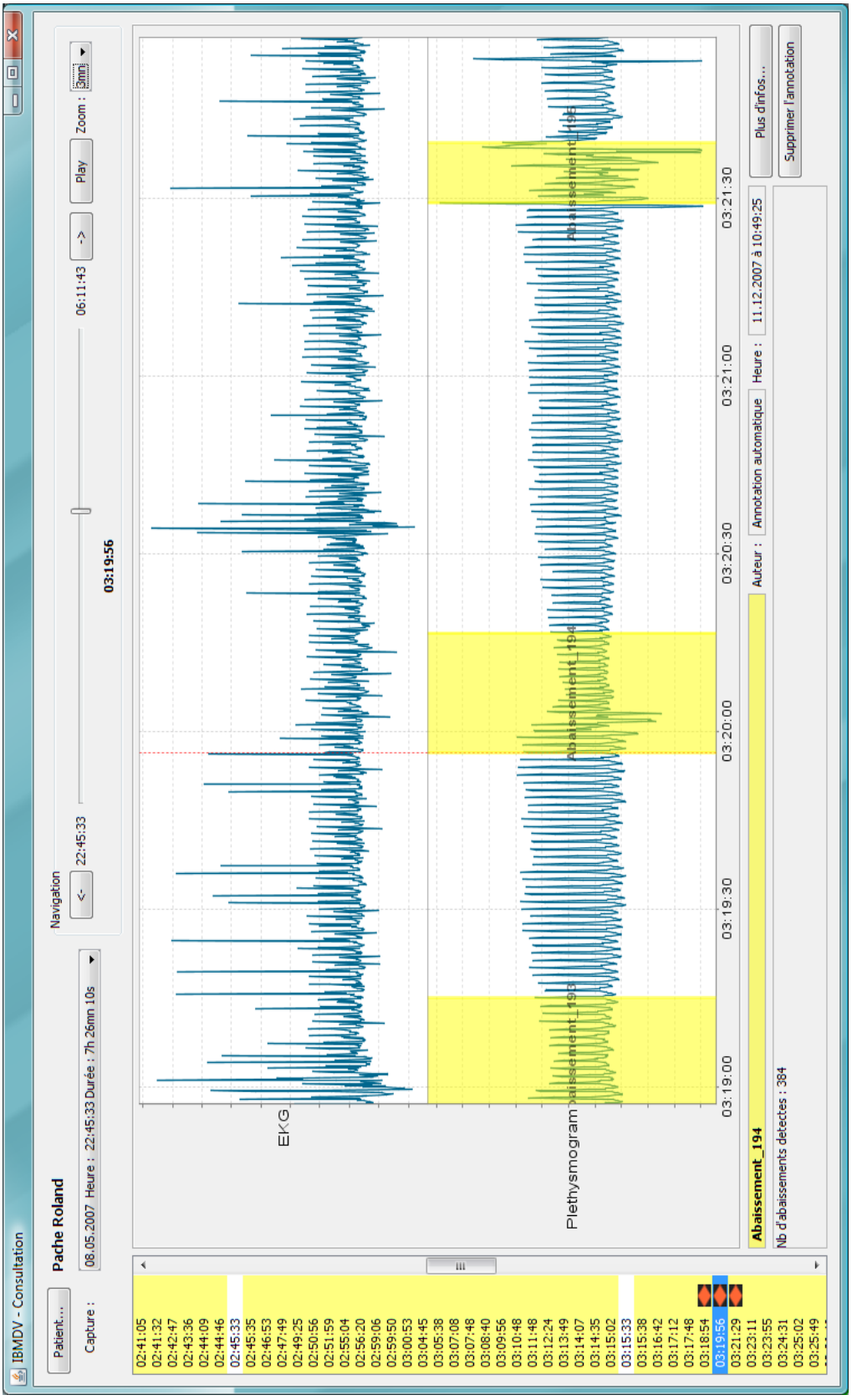


FIG. B.3 – Printscreen n°3

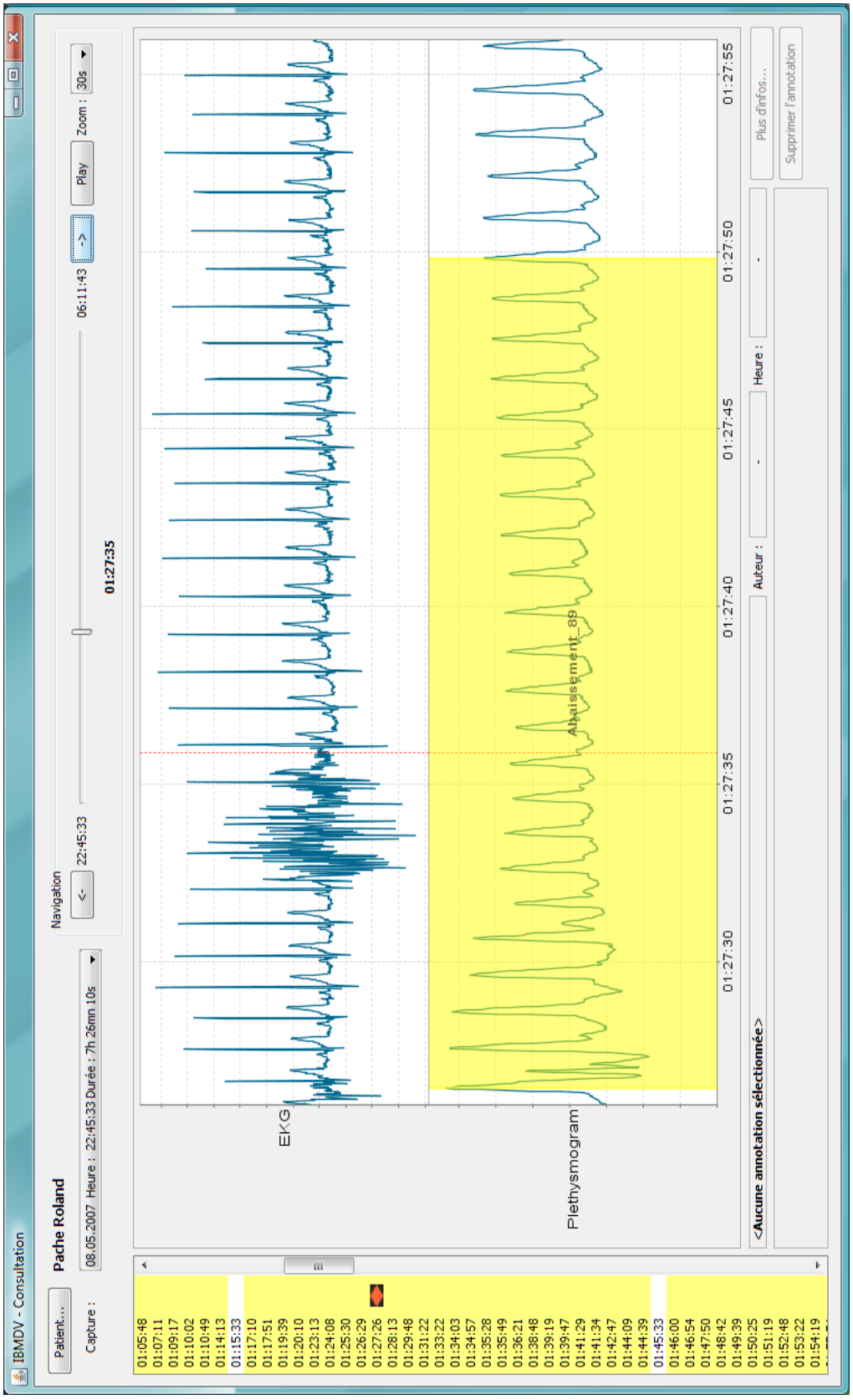


FIG. B.4 – Printscreen n°4

Annexe C

Journal de travail

C.1 Semaine du 18 au 21 septembre

Voici les points traités :

- Discussion avec Mme Raileanu, M. Poulin, M. Jaton et M. Gavin pour trouver un format d'échange entre le deux projets.
- Début de la mise en place d'une interface, pour le programme final.
- Début de la rédaction du rapport sous LaTeX (journal de travail, cahier des charges, caneva rapport).
- Tests concernant la sauvegarde de graphique réalisé par Rserve, dans un fichier image (.png, .pdf, etc).
- Mise en place d'un format XML avec M. Gavin, pour définir précisément les informations à transmettre entre les deux projets.

C.2 Semaine du 24 au 28 septembre 2007

Voici les points traités :

- Réalisation de la partie concernant le transfert du résultat final du programme.
- Tests et discussion avec M. Poulin pour la sauvegarde de graphique réalisé par Rserve, dans un fichier image.
- Réussite de la sauvegarde d'un graphique réalisé avec Rserve, dans un fichier postscript (Rplots.ps).

C.3 Semaine du 1 au 5 octobre 2007

Voici les points traités :

- Tests pour la copie du fichier Rplot.ps, pour générer d'autre graphiques sans supprimer les précédents. Problème lors de la copie...
- Test du modèle statistique ARIMA et récupération des différents résultats.
- Mise au propre du code déjà réalisé (commentaire, finition, etc)

C.4 Semaine du 8 au 12 octobre 2007

Voici les points traités :

- Tests du modèle ARIMA, et discussion avec Mme Raileanu pour la suite du projet.
- Rédaction du rapport final, en tenant compte des remarques du dossier du pré-projet.

C.5 Semaine du 15 au 19 octobre 2007

Voici les points traités :

- Remodélisation de l'interface graphique.
- Mise en place des sauvegardes des différents résultats (fichier XML, fichier HTML).

C.6 Semaine du 22 au 26 octobre 2007

Voici les points traités :

- Mise en place d'une console pour exécuter différents plugins disponibles.
- Finitions des sauvegardes des différents résultats (fichier XML, fichier HTML).

C.7 Semaine du 29 octobre au 2 novembre 2007

Voici les points traités :

- Rédaction du rapport final.
- Rendez-vous avec Mme Raileanu pour discuter de l'avancement du projet.
- Une visite du centre du sommeil du CHUV est prévu grâce à M Jaton et le Dr Heinzer, le vendredi 9 novembre.
- Finition du programme pour le présenter au centre du sommeil.

C.8 Semaine du 5 au 9 novembre 2007

Voici les points traités :

- Rédaction du rapport final.
- Journée du forum HES-SO 07.
- Visite du centre du sommeil du CHUV.

C.9 Semaine du 12 au 16 novembre 2007

Voici les points traités :

- Rédaction du rapport final.
- Relecture intermédiaire du rapport final.
- Tests pour la détection du l'abaissement du pouls d'un patient.

C.10 Semaine du 19 au 23 novembre 2007

Voici les points traités :

- Recherches des fonctions disponible par R pour l'analyse du pouls.
- Rendez-vous avec M. Poulin pour discuter des améliorations possibles.
- Améliorations graphiques apportées au projet.
- Début du développement de l'analyse pour le CHUV.
- Rédaction du rapport final.

C.11 Semaine du 26 au 30 novembre 2007

Voici les points traités :

- Rendez-vous avec Mme Raileanu pour discuter de l'analyse du pouls.
- Avancement de l'analyse pour le CHUV.
- Discussion avec M. Gavin pour voir le résultat de l'analyse pour le CHUV, et régler les derniers problèmes de compatibilité.
- Affichage du résultat de l'analyse du CHUV, grâce à jfreeChart.
- Rédaction du rapport final, plus relecture.

C.12 Semaine du 3 au 7 décembre 2007

Voici les points traités :

- Avancement de l'analyse pour le CHUV.
- Rédaction du rapport final.

C.13 Semaine du 10 au 15 décembre 2007

Voici les points traités :

- Rédaction du rapport final.
- Relecture complète du rapport.